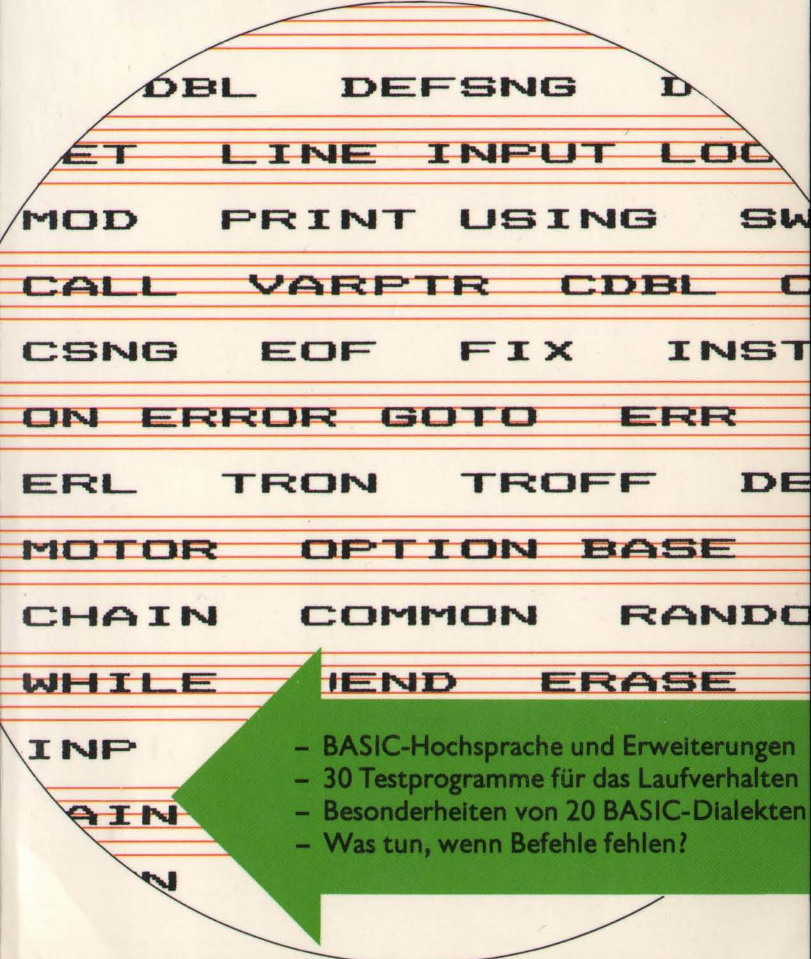
humboldt

BASIC-Dialekte



- BASIC-Hochsprache und Erweiterungen
- 30 Testprogramme für das Laufverhalten
- Besonderheiten von 20 BASIC-Dialekten
- Was tun, wenn Befehle fehlen?

BASIC-Dialekte

humboldt
Ratgeber

Von demselben Verfasser:

Mikroprozessoren – Kleinstcomputer	ht 338
Tischcomputer für Heim + Beruf	ht 415
Programmiersprache BASIC – Schritt für Schritt	ht 456
BASIC für Fortgeschrittene	ht 496
Bausteine für BASIC-Programme (ab 9/88)	ht 591

In der gleichen Reihe:

Datenverarbeitung – verständlich gemacht	ht 200
Basteln mit elektronischen Bauelementen	ht 414
Video	ht 422
Bildschirmtext	ht 457
Schachcomputer	ht 465
Lernen mit dem Homecomputer. BASIC-Programme für Schularbeiten	ht 525
Spielend Programmieren lernen	ht 526
Programmiersprache PASCAL – Schritt für Schritt	ht 551
Elektronik-Basteln für Fortgeschrittene	ht 560
So finde ich den richtigen Computer	ht 564

Weitere Titel in Vorbereitung

BASIC-Dialekte

Von Dr. Hans-Joachim Sacht

humboldt

humboldt-taschenbuch 524

Umschlaggestaltung: Christa Manner, München
Die Vorlage für die Kreisabbildung auf dem Broschurumschlag stellte uns der Autor zur Verfügung.
Ebenfalls vom Autor stammen alle Vorlagen der Textabbildungen im Innenteil des Bandes sowie alle Schema-Vorlagen.
Reinzeichnungen der Grafiken auf den Seiten 29, 30, 33, 43, 46, 50, 169, 170, 171, 180: Fritz E. Urich, München.
Wir danken den folgenden Firmen für ihre Unterstützung, insbesondere für Überlassung ihrer Computer-Handbücher und diverser Informationsmaterialien:
Hewlett Packard, Bad Homburg
IBM Deutschland GmbH, Stuttgart
ACORN Computers International Ltd., München
CE-TEC Trading GmbH, Hamburg
Commodore GmbH, Frankfurt
Triumph-Adler AG, Nürnberg
MVB GmbH, Ebersburg-Weyer
Casio Computer Co. GmbH Deutschland, Hamburg
ATARI ELEKTRONIK Vertriebsges. mbH, Hamburg
Apple Computer GmbH, München
Jürgen Schumpich GmbH, Ottobrunn
Sharp Electronics GmbH, Hamburg
Epson Deutschland GmbH, Düsseldorf
Tandy Corporation Germany, Ratingen
Bernd Jöllenbeck GmbH, Weertzen
Microsoft GmbH, Taufkirchen
Schneider Computer, Türkheim
Texas Instruments Deutschland GmbH, Freising
Deutsche Thomson-Brandt GmbH, Bremen
TCS Computer GmbH, St. Augustin

Hinweis für den Leser:
Alle Angaben – ohne Gewähr – entsprechen dem Stand von 1987. Sie sind von Autor und Verlag sorgfältig geprüft. Dennoch kann eine Gewähr nicht übernommen werden.

Druck: Presse-Druck Augsburg
Printed in Germany
ISBN 3-581-66524-7

23 * 87

Inhalt

Vorwort

Die weite Verbreitung der Programmiersprache BASIC hat dazu beigetragen, daß heute unzählige BASIC-Dialekte bestehen. Es gibt kaum noch einen Computer, der sich mit dem von der amerikanischen Normbehörde ANSI festgelegten MIN STANDARD begnügt. Praktisch alle neuen BASIC-Versionen erlauben auch Zeichenkettenverarbeitung und Fließkommarechnung. Auch mehr oder minder komfortable Befehle für die Erstellung von Grafiken und für die Tonerzeugung sind heute üblich.

Leider werden im erweiterten BASIC nicht immer die gleichen Befehlswörter verwendet, und nicht alle heute üblichen Zusatzbefehle laufen auf allen Computern. Oft ist auch noch die Wirkung ähnlicher Befehle auf verschiedenen Computern unterschiedlich, auch wenn das gleiche Befehlswort verwendet wird.

Wer Grundkenntnisse in BASIC erworben und die ersten kleinen Programme selbst geschrieben hat, wird bald den Wunsch hegen, auch an größere Aufgaben heranzugehen. Hierfür braucht er Vorbilder, und die findet er in Büchern und Fachzeitschriften; nur muß er sie auch verstehen und nachvollziehen können.

Dieses Buch will helfen, indem es die wesentlichen Unterschiede der einzelnen BASIC-Dialekte erklärt und Hinweise gibt, wie man das eigene BASIC prüfen und gegebenenfalls dort nicht vorhandene Befehle ersetzen kann.

Obwohl es kaum zwei von verschiedenen Computerherstellern angebotene BASIC-Versionen gibt, die sich in allen Einzelheiten gleichen, hat sich doch eine Art »BASIC-Hochsprache« herausgebildet, die auf (fast) allen Computern läuft. Sie wird im 1. Kapitel behandelt.

Alle neueren BASIC-Dialekte sind aber weiter ausgebaut. Die bei den erweiterten BASIC-Versionen üblichen Befehle werden im 2. Kapitel erklärt.

Wenn auch bei den Grafik- und Tonbefehlen vielfach die gleichen Befehlswörter verwendet werden, so sind die Möglichkeiten, die dem Programmierer auf diesen Gebieten geboten werden, doch sehr unterschiedlich. Der prinzipielle Aufbau dieser Befehle wird im 3. Kapitel behandelt.

Da nicht nur die Befehle, sondern auch ihre Wirkung, d. h. das Laufverhalten, unterschiedlich ausfallen können, zeigen wir im 4. Kapitel eine Reihe von Tests, mit denen jeder das Laufverhalten seines Computers überprüfen kann.

Meist werden in Programmen aber noch weitere computerspezifische Befehle verwendet. Reine Übersetzungstabellen für die Wandlung der Befehle einer Version in eine andere sind unübersichtlich und mühsam zu benutzen. Im 5. Kapitel, in dem wir die besonderen Eigenschaften von 20 viel verwendeten BASIC-Dialekten erklären, beschreiten wir darum einen anderen Weg.

Ausgehend von den Befehlen der BASIC-Hochsprache und von den wichtigsten Befehlen des erweiterten BASIC wird nur das in den einzelnen Abschnitten behandelt, was vom Grundstock abweicht. Daß hierbei allerdings nicht alle Einzelheiten erschöpfend erklärt werden können, ergibt sich aus dem Umfang eines Taschenbuchs.

In den weiteren Kapiteln geben wir konkrete Hinweise, wie man fremde Programme verstehen und in die eigene BASIC-Version konvertieren kann. Hierzu gehört der Ersatz fehlender Befehle ebenso wie die Erklärung von Befehlsverwendungen, die nicht auf den ersten Blick verständlich sind.

Bei der Komplexheit der Materie kann dieses Buch keine »Kochrezepte« anbieten, wie man Programme umstellt. Es gibt aber eine Fülle von Hinweisen und Anregungen, wie man dabei vorgehen kann.

Vor allem soll dieses Buch helfen, auf fremden Computern entwikkelte Programme zu verstehen, um daraus zu lernen und um diese auf dem eigenen Computer (vielleicht in noch verbesserter Form) selbst zu gestalten.

Nicht zuletzt wollen wir durch dieses Buch alle, die Programme veröffentlichen wollen, dazu anregen, ihre Programme so einfach wie möglich mit den allgemein verwendeten Befehlen zu schreiben, damit auch andere Computerfreunde davon Nutzen haben können.

Dr. Hans-Joachim Sacht

1. Die Befehle der BASIC-Hochsprache

Von der amerikanischen Normbehörde wurde vor einigen Jahren eine Normung für BASIC vorgeschlagen, die als ANSI MIN STANDARD bekannt wurde. Sie reicht einerseits heute für die universelle Verwendung von BASIC nicht mehr aus, enthält anderseits aber einige Befehle, die in moderne Versionen nicht aufgenommen wurden. Vor allem gehören die im ANSI MIN STANDARD nicht enthaltenen Befehle für die Behandlung von Zeichenketten heute zu den bei allen Versionen vorhandenen Grundbefehlen.

Bei dieser Gelegenheit muß zunächst noch eine im Deutschen entstandene Sprachverwirrung geklärt werden. Die Worte »Befehl«, »Kommando« und »Anweisung« werden leider in der deutschsprachigen Computerliteratur nicht eindeutig verwendet. In diesem Buch wird *Befehl* für alle BASIC-Schlüsselwörter gebraucht, die den Computer zu einer bestimmten Arbeit veranlassen. Ein Befehl kann dementsprechend ein direktes *Kommando* sein, das der Computer sofort nach der Eingabe ausführt, oder eine *Anweisung*, die im Programm steht und erst zur Verarbeitung kommt, wenn die betreffende Zeile beim Programmablauf an der Reihe ist.

Die Tabelle Abb. 1 zeigt die Befehle, die bei allen modernen, voll ausgebauten BASIC-Versionen verfügbar sind. Man kann mit diesen Befehlen schon praktisch alle Programmieraufgaben (ohne Dateiverwaltung, Grafik und Tonerzeugung) lösen. Programme, die nur unter Benutzung dieser Befehle geschrieben wurden, sind gut verständlich. Sie laufen mit geringfügigen Anpassungen auf allen Computern. Man kann also mit Recht behaupten, daß sich hier eine Art »BASIC-Hochsprache« herausgebildet hat.

Die Tabelle enthält keine Befehle für die Bedienung von externen Speichern und Druckern, die natürlich in allen BASIC-Versionen enthalten sein müssen. Es existieren also stets weitere Befehle. Sie

Die wichtigsten BASIC-Befehle

Komman-dos	RUN	Startet ein Programm		
	LIST	Gibt Programmliste aus		
	NEW	Löscht Programmspeicher		
	CONT	Fortsetzung Programmlauf		
Anweisungen	PRINT....	Ausgabebefehl		
	LET=...	Wertzuweisung an Variable		
	INPUT.,..,.	Aufforderung zur Dateneingabe		
	READ.,..,.	Einlesen von Daten		
	DATA.,..,.	Daten im Programm		
	GOTO...	Unbedingter Sprung		
	FOR..=..TO..STEP	Schleifendefinition		
	NEXT..	Schleifenende		
	IF...THEN...ELSE	Bedingter Befehl		
	GOSUB...	Anruf Unterprogramm		
	ON...GOTO...	Errechneter Einsprung		
	RETURN	Rücksprung zum Hauptprogramm		
	DIM..(.,.)	Speicherplatzreservierung		
	STOP	Anhalten des Programmlaufs		
	END	Ende des Programms		
	REM	Für Kommentarzeilen		
Operatoren	+	Addition	=	Gleichheit
	−	Subtraktion	<>	ungleich
	*	Multiplikation	>	größer als
	/	Division	>=	größer oder gleich
	∧	Potenzierung	<	kleiner als
	(.)	Vorrang	<=	kleiner oder gleich
	AND OR NOT		Logische Verknüpfungen	
Numerische Funktionen	ABS (X) Absolutwert		SQR (X) Quadratwurzel	
	SGN (X) Vorzeichen		EXP (X) e^x	
	INT (X) Ganze Zahl		LOG (X) Logarithmus	
	SIN (X) Sinus		LN (X) Natürl. Logarithmus	
	COS (X) Cosinus		RND (Ø) Zufallszahl	
	TAN (X) Tangens		TAB (X) Tabulatorstellen	
	ATN (X) Arcustangens		SPC (X) Zwischenräume	
Zeichenketten Funktionen	LEN (X$)	Länge der Zeichenkette		
	ASC (X$)	ASCII Code des 1. Zeichens, dezimal		
	CHR $ (X)	Zeichen entsprechend ASCII dezimal		
	VAL (X$)	Wandelt Ziffernkette in Zahl		
	STR $ (X)	Wandelt Zahl in Ziffernkette		
	LEFT $ (X$, n)	Trennt die n vorderen Zeichen ab		
	RIGHT $ (X$, n)	Trennt die n hinteren Zeichen ab		
	MID $ (X$, m, n)	Trennt vom mten Zeichen n Zeichen ab		

Abb. 1: Die Befehle der BASIC-Hochsprache

weichen aber viel deutlicher voneinander ab als die Grundbefehle. Ganz besonders gilt dies für die Grafikbefehle und für die Befehle zur Tonerzeugung, deren Wirkung bei den verschiedenen Dialekten kaum noch vergleichbar ist.

Leider bestehen aber auch bei den Befehlen der BASIC-Hochsprache noch Abweichungen in den einzelnen Dialekten, die bei der Konvertierung von Programmen Beachtung finden müssen. Im folgenden wird darum der Wortschatz der BASIC-Hochsprache genau erklärt und auf mögliche Unterschiede hingewiesen.

1.1 Kommandos

Die größte Übereinstimmung herrscht bei den *Kommandos*, also bei den Befehlen, die der Computer im Direktbetrieb verarbeitet.

RUN Das Kommando für den Programmstart ist fast immer mit nachgestellter Zeilennummer zulässig. Alle Variablenspeicher werden gelöscht. Einige BASIC-Versionen erlauben die Verwendung dieses Kommandos auch als Programm-Anweisung, was für die Koppelung mehrerer Programme nützlich sein kann.

LIST Bei dem Kommando für die Darstellung des Programms auf dem Bildschirm ist immer auch eine teilweise Ausgabe möglich. Bei einigen Dialekten muß ein Minuszeichen zur Trennung der Zeilennummern, also zum Beispiel **LIST 100–200**, geschrieben werden. Andere verwenden das Komma. Man muß dann schreiben **LIST 100,200.**

NEW Dieses Kommando wird immer benutzt, um das im Arbeitsspeicher stehende Programm zu löschen. Bei einigen Versionen wird hierfür **SCR** verwendet. Im allgemeinen wird aber das Programm nicht physikalisch gelöscht, sondern nur der auf den Programmanfang gerichtete Zeiger. Es gibt manchmal besondere Befehle oder auch Tricks, mit denen ein versehentlich gelöschtes Programm noch gerettet werden kann.

CONT Mit diesem Kommando wird der Lauf eines mit **BREAK** oder **STOP** abgebrochenen Programms fortgesetzt, wenn keine Änderung erfolgt ist.

1.2 Operatoren

Die *Operatoren* dienen in einem BASIC-Ausdruck zur Verbindung der dort stehenden Konstanten oder Variablen. Oft unterscheidet man zwischen *Rechenoperatoren, Vergleichsoperatoren* und *logischen Operatoren* (englisch BOOLEAN OPERATOR). Fast keine Abweichungen gibt es bei den Rechenoperatoren. Bei den Vergleichsoperatoren und bei den logischen Operatoren registrieren wir jedoch Abweichungen in der Wirkung, da intern unterschiedlich gearbeitet wird.

+ –	Bei den Operatoren für die Addition und die Subtraktion gibt es keine dialektspezifischen Unterschiede.
* /	Der Stern für die Multiplikation und der Schrägstrich für die Division werden allgemein verwendet.
( )	Klammern begründen einen Vorrang bei Rechenoperationen. Im übrigen gilt die Regel: Punkt vor Strich. Grundsätzlich wird nach dem AOS-Verfahren und nicht entsprechend UPN gerechnet.
↑ ^	Als Operator für die Potenzierung dient eigentlich der Aufwärtspfeil mit dem dezimalen ASCII-Code 94. Da die Code von 91 bis 94 aber zum Beispiel bei der deutschen ASCII-Version für die großen Umlaute verwendet werden, ist die Darstellung des Operators für die Potenzierung manchmal etwas verwirrend. Beim TRS 80 wird eine eckige Klammer ([), bei Genie der große Buchstabe Ä angezeigt bzw. gedruckt. Bei einigen Versionen, nicht zuletzt dem PC 100, sind zwei hintereinander stehende Sterne (**) zu benutzen.
= < > > = < =	Auch als Vergleichsoperatoren werden einheitliche Symbole verwendet. Trotzdem gibt es hier bemerkenswerte Unterschiede bei der Wirkung im Befehl. Vielfach wird – 1 bei richtigem Vergleich und 0 bei unrichtigem Vergleichsergebnis erzeugt. Es gibt aber auch Dialekte, bei denen als Kennung für einen wahren Vergleich +1 gemeldet wird.
	Beim Vergleich von Buchstaben wird der ASCII-Codewert zugrunde gelegt, bei dem der Wert für A

kleiner ist als der für B und so fort. Während das Gleichheitszeichen allgemein verarbeitet wird, erlauben nicht alle BASIC-Versionen die Verwendung der Zeichen für »größer als« und »kleiner als« beim Vergleich von Buchstaben.

<> >< Geringe Unterschiede registrieren wir bei dem Ungleich-Operator. Meist werden zwei gegeneinandergestellte Winkel verwendet. Einige Versionen gebrauchen aber das Doppelkreuz (#).

AND
OR
NOT
Die Operatoren für die logische Verknüpfung sind bei allen BASIC-Versionen gleich. Es ist üblich, daß die vor und nach den Operatoren stehenden Zahlen zu Binärzahlen gewandelt und dann bitweise verarbeitet werden. Einige Computer geben das Ergebnis auch wieder als Zahl aus, andere behandeln die Argumente nur als »wahr« oder »falsch«.

\+ & Das mathematische oder – bei manchen Versionen – das kaufmännische Additionszeichen wird im allgemeinen als Operator für das Zusammenfügen von Zeichenketten ohne Zwischenraum verwendet.

1.3 Satz- und Sonderzeichen

Satz- und Sonderzeichen spielen bei allen Programmiersprachen eine große Rolle. BASIC verwendet nur die Zeichen, die praktisch auf allen Schreibmaschinentastaturen amerikanischer Bauart vorhanden sind. Es gibt hier Besonderheiten bei einigen BASIC-Versionen, die nicht in den USA entstanden sind. Die amerikanischen Versionen behandeln zwar die Bedeutung der Zeichen im wesentlichen gleich; trotzdem ergeben sich unterschiedliche Wirkungen bei einigen Zeichen und Befehlen.

. Statt des bei uns üblichen Kommas wird immer der Punkt zur Dezimaltrennung verwendet.

, Das Komma dient meist als Trennzeichen zwischen einzugebenden, zu druckenden, zu speichernden oder zu lesenden Ausdrücken beziehungsweise Variablen. Es gibt aber recht unterschiedliche Wirkungen, je nachdem, in welchen Befehlen es benutzt wird.

; Auch das Semikolon dient als Trennzeichen in einigen Befehlen. Oft kann es zusätzlich am Ende einer

Druckzeile verwendet werden, um den Zeilenvorschub zu verhindern.

: Der Doppelpunkt wird bei den meisten BASIC-Versionen benutzt, um mehrere Befehle in einer Zeile voneinander zu trennen. Es gibt aber auch Versionen, die hierfür das Semikolon oder den Schrägstrich vorschreiben.

» Anführungsstriche werden immer zur Kennzeichnung von Zeichenketten verwendet. Meist dürfen sie darum im Inneren einer Kette nicht vorkommen. Bei einigen Versionen können die abschließenden Anführungsstriche am Ende einer Programmzeile fehlen.

$ Allgemein wird das Dollarzeichen zur Kennzeichnung von Zeichenkettenvariablen verwendet. Der in Schweden entwickelte Computer ABC 80 verwendet statt dessen das schwedische Währungszeichen.

% Das Prozentzeichen dient als Kennzeichen für Ganzzahl-Variable.

Das Doppelkreuz (kaufmännisches Nummer-Zeichen) dient bei einigen Computern zum Kennzeichnen von Variablen mit doppelter Genauigkeit. Es wird auch bei Kassetten- und Diskettenbefehlen als Numerierungszeichen für die Kanal- oder Dateinummer verwendet.

! Das Ausrufungszeichen kennzeichnet bei einigen Computern Variable mit einfacher Genauigkeit.

? Das Fragezeichen wird allgemein als Kurzeingabe für den PRINT-Befehl anerkannt. Auf dem Bildschirm bedeutet es die Aufforderung zu einer Eingabe nach einem INPUT-Befehl.

@ Das kaufmännische Zeichen für at (Klammeraffe) wird beim Genie und beim TRS 80 nach dem PRINT-Befehl verwendet, wenn die Bildschirmausgabe wie beim PRINT AT-Befehl an eine bestimmte Stelle geschrieben werden soll. Auf Druckern mit deutschem Zeichensatz wird es oft als § (Paragraph) gedruckt.

' Das Hochkomma (Apostroph) kann bei einigen Versionen zur Kennzeichnung von Kommentaren be-

nutzt werden. Es sind aber auch andere Satzzeichen üblich.

& Das kaufmännische Und-Zeichen wird bei einigen Dialekten als Vorzeichen für nicht-dezimale Zahlenstellung verwendet.

1.4 Funktionen

Funktionen benötigen immer ein (meist in einer Klammer stehendes) *Argument.* Es gibt BASIC-Versionen mit 50 und mehr verschiedenen Funktionen. Für die BASIC-Hochsprache genügt eine Reihe von Grundfunktionen, aus denen alle anderen abgeleitet werden können. Bei einigen Funktionen gibt es aber auch in der Hochsprache schon geringfügige Abweichungen.

Numerische Funktionen mit Zahlen als Argument

SIN
COS
TAN
ATN
Für trigonometrische Rechnungen sind immer die Funktionen für den Sinus, den Cosinus, den Tangens und den Arcustangens verfügbar. Allgemein muß der Winkel im Bogenmaß als Argument in die Funktion eingesetzt werden. Nur wenige Versionen erlauben die Eingabe von Winkelgraden nach vorangestelltem Grad-Befehl.

ABS Die Funktion zur Bestimmung des Absolutwertes einer Zahl ist immer im Befehlssatz enthalten.

SGN Auch die Funktion, die das Vorzeichen einer Zahl ausgibt, ist immer verfügbar, obwohl sie leicht durch einen Vergleich mit 0 ersetzt werden könnte.

SQR Speziell für die Berechnung der Quadratwurzel ist eine Funktion vorhanden. Andere Wurzeln können durch Potenzieren mit gebrochenem Exponenten berechnet werden.

LOG Im allgemeinen wird mit der Logarithmus-Funktion der natürliche Logarithmus zur Grundzahl e berechnet.

EXP Mit der EXP-Funktion wird die Zahl berechnet, die sich ergibt, wenn das Argument als Hochzahl zur Zahl e verwendet wird.

INT Diese Funktion rundet eine gebrochene Zahl auf die nächstkleinere Ganzzahl ab. Sie arbeitet also bei negativen Zahlen etwas anders als die auch vielfach vorhandene **FIX**-Funktion, bei der immer die Nachkommastellen abgeschnitten werden.

RND Bei der Funktion zur Erzeugung von Zufallszahlen gibt es erhebliche Dialekt-Unterschiede bezüglich des einzusetzenden Arguments. Einige Varianten werden in den Kapiteln 4 und 5 erklärt.

TAB Nach der Tabulator-Funktion darf in vielen BASIC-Versionen kein Zwischenraum vor der Klammer stehen.

SPC Mit dem Argument in der Klammer wird die Anzahl von Leerstellen, die in einem PRINT-Befehl eingesetzt werden sollen, bestimmt. Einige BASIC-Versionen verwenden zu diesem Zweck die Funktion **STRING$ (a,b)**, mit der a-mal die Wiederholung des als ASCII-Wert eingegebenen Zeichens b zu programmieren ist.

FRE Beim Aufruf dieser Funktion melden viele Computer, je nach Argument, den gesamten freien Speicherraum oder den freien Raum zur Speicherung von Zeichenketten. Vielfach wird für den ersten Zweck auch **MEM** verwendet.

Funktionen zur Wandlung von Zeichenketten

LEN Die Funktion zur Ausgabe der Länge einer Zeichenkette ist in allen Versionen vorhanden, die Zeichenkettenbearbeitung zulassen.

LEFT$
RIGHT$ Beide Funktionen dienen zum Abtrennen einer vorgegebenen Anzahl von Zeichen vom linken oder rechten Ende der Kette. Sie sind nicht in allen Versionen verfügbar.

MID$** Zum Aussondern eines oder mehrerer Zeichen in einer Kette brauchen wir die MID$-Funktion, die in einigen Versionen auch **SEG$ heißt.

CHR$ Mit dieser recht oft verwendeten Funktion wird das ASCII-Zeichen erzeugt, dessen Code-Wert als Dezimalzahl im Argument steht.

ASC Die Funktion stellt den dezimalen ASCII-Wert des ersten Zeichens der im Argument folgenden Zeichenkette bereit.

VAL Diese Funktion dient zur Wandlung einer aus Ziffern bestehenden Zeichenkette in eine Zahl, mit der gerechnet werden kann.

STR$ Im Gegensatz zu VAL wandelt STR$ eine Zahl in eine Ziffernkette um.

1.5 Programmanweisungen

Die wichtigsten BASIC-Befehle sind die Programm-Anweisungen (Statements). Sie stehen im Programmbetrieb in Befehlszeilen, die der Computer an der vorangestellten Zeilennummer erkennt. Viele BASIC-Versionen erlauben die Verwendung der meisten Anweisungen auch als Kommandos im Direktbetrieb. Im übrigen gibt es in dieser Befehlsgruppe nur geringe Dialekt-Abweichungen, die aber doch bei einer Konvertierung zu beachten sind.

DATA Einige Computer erlauben keine über mehrere Zeilen gehende DATA-Statements. Unterschiedlich ist auch die Schreibweise von Zeichenketten in diesem Befehl. Bei einigen Versionen können die Anführungszeichen am Anfang und am Ende eines Strings entfallen, wenn dieser nicht mit einem Leerzeichen beginnt.

DIM Bei den meisten Dialekten brauchen nur Felder mit mehr als 10 oder 11 Elementen vorher dimensioniert zu werden. Es gibt aber auch BASIC-Versionen, die eine Dimensionierung der Länge von Zeichenketten vorschreiben.

END Meistens ist der End-Befehl nicht erforderlich, wenn das Programm mit der letzten gespeicherten Programmzeile endet.

FOR...TO Der Schleifenbefehl kann in einzelnen Dialekten unterschiedlich laufen, wenn die Bedingung für den Abbruch schon am Schleifenanfang erfüllt ist. Bei den meisten Dialekten erfolgt dann trotzdem ein Schleifendurchgang; bei wenigen wird die Schleife übersprungen.

GOSUB Der Befehl zum Ansprung eines Unterprogramms ist bei allen BASIC-Versionen gleich. Unterschiede bestehen lediglich hinsichtlich der Anzahl der erlaubten Verschachtelungen, das heißt, der noch nicht abgeschlossenen Unterprogramme. Bei einigen Versionen kann man als Sprungziel statt der Zeilennummer einen Label-Namen angeben.

GOTO Auch der unbedingte Sprungbefehl ist überall gleich. Er wird oft in zwei Wörtern **GO TO** geschrieben. Wenige Computer erlauben Rechenausdrücke statt einer vorgegebenen Zeilennummer als Sprungziel. Gelegentlich sind auch Label-Namen zugelassen.

IF...THEN Beim Entscheidungsbefehl, der eine bedingte Verzweigung veranlaßt, gibt es einige Unterschiede. Meist kann nach THEN sofort die Zeilennummer des Sprungziels bei erfüllter Bedingung stehen. Oft wird auch GOTO statt THEN angenommen. Häufig können nach THEN ein oder mehrere beliebige andere Befehle folgen. Das Befehlswort **ELSE**, dem Befehle folgen, die bei nicht erfüllter Bedingung zu durchlaufen sind, ist nicht bei allen Dialekten zulässig. Es gibt Dialekte, bei denen THEN durch ein Komma ersetzt werden kann.

INPUT Mit diesem Befehl kann man eine Tastatureingabe einer Variablen zuweisen. Die meisten Computer verweigern die Annahme einer Eingabe, wenn versucht wird, Buchstaben einer Zahlenvariablen zuzuordnen. Meist wird der Benutzer aufgefordert, die Eingaben zu wiederholen, ohne daß das Programm zum Abbruch kommt. Bei einigen BASIC-Versionen wird das Programm aber abgebrochen, wenn die RETURN-Taste ohne vorherige Betätigung einer anderen Taste gedrückt wird.

Mehrere durch Kommas getrennte Eingaben sind vielfach zugelassen. Alle neueren Versionen erlauben auch, daß in den Eingabebefehl ein erklärender Text aufgenommen wird, der dann wie bei einem PRINT-Befehl ausgegeben wird.

LET= Bei fast allen BASIC-Versionen kann das Befehlswort LET bei Zuweisungsbefehlen auch entfallen. Einige Computer melden einen Fehler, wenn im Programm

eine Variable angesprochen wird, der vorher noch kein Wert zugewiesen wurde. Andere setzen dann bei Zahlenvariablen 0 und bei Zeichenkettenvariablen einen Leerstring ein.

NEXT Dieses Befehlswort wird immer am Ende einer Schleife verwendet. Meist braucht keine Variable eingesetzt zu werden, wenn nur eine einfache, unverschachtelte Schleife läuft. In einigen Versionen können dem Befehlswort auch mehrere Variable folgen, durch Kommas getrennt, und zwar dann, wenn mehrere Schleifen gemeinsam zum Ende kommen.

ON ..GOTO ON ..GOSUB Die Befehle für Mehrfachverzweigungen sind in allen BASIC-Versionen praktisch gleich. Geringe Unterschiede kann es bei der Anzahl der erlaubten Verzweigungen geben und bei der Verarbeitung von falschen Eingaben für die Entscheidungsvariable.

PRINT Der Ausgabebefehl kann praktisch immer bei der Tastatureingabe durch das Fragezeichen ersetzt werden. In die Programmliste wird dann meist selbständig das Befehlswort PRINT eingesetzt. Als Trennzeichen sind fast immer das Komma und das Semikolon zulässig. Kommas bewirken die Ausgabe in Kolonnen. Semikolons ergeben bei Zeichenketten eine Darstellung ohne Zwischenräume. Bei Zahlen wird meist eine Freistelle hinter die Ziffern gesetzt und oft auch die Stelle vor den Ziffern freigelassen, wenn bei positiven Zahlen kein Vorzeichen gesetzt wird.

READ Der Befehl zum Einlesen von Daten aus DATA-Zeilen ist praktisch überall gleich.

REM Mit diesem Kürzel können Kommentare im Programm gekennzeichnet werden, die der Computer im Programmlauf nicht beachtet. Bei einigen Computern genügt hierfür auch das Apostroph ('). Auch andere Kurzzeichen für diesen Befehl werden verwendet.

RESTORE veranlaßt den READ-Befehl, wieder Daten von der ersten Datenzeile ab zu lesen. Zulässig ist auch manchmal die Angabe der Zeilennummer, von der an erneut gelesen werden soll.

RETURN Dieser Befehl muß immer am Ende eines Unterprogramms stehen und bewirkt dann den Rücksprung zum Befehl nach dem Anruf des Unterprogramms.

STEP Die Veränderung der Laufvariablen in FOR ..NEXT-Schleifen braucht allgemein nicht programmiert zu werden, wenn sie in Schritten von 1 erhöht werden soll. Bei einigen BASIC-Versionen dürfen nur Ganzzahlen hinter STEP eingesetzt werden.

STOP Der Befehl zum Anhalten eines laufenden Programms lautet in einigen BASIC-Versionen **BREAK.**

1.6 Befehlsschreibung

Normalerweise werden die BASIC-Befehlswörter mit Großbuchstaben geschrieben. Einige Dialekte wandeln in Kleinbuchstaben eingegebene Befehle im Ausdruck selbsttätig in Großbuchstaben um. Es gibt allerdings auch Versionen, bei denen zwei verschiedene Zeichensätze schaltbar sind. In diesem Fall müssen oder können Befehlswörter klein geschrieben werden und bleiben im Listing auch stehen.

Das Lesen eines Programms wird sehr erleichtert, wenn vor und hinter den BASIC-Befehlswörtern eine Leerstelle bleibt. Es gibt Dialekte, bei denen dies sogar Vorschrift ist. Andere setzen die Leerstellen automatisch oder erlauben die Befehlsschreibung mit und ohne Leerstellen.

Die Länge der Befehlszeilen ist immer begrenzt. Zwischen 80 und ca. 255 Zeichen sind üblich. Meist können mehrere Befehle, durch Doppelpunkte getrennt, in eine Zeile geschrieben werden, wobei die Länge der Bildschirmzeile keine Rolle spielt.

Da die BASIC-Befehle intern im Speicher immer als Code in einem Byte, genannt *Token*, abgelegt werden, bieten viele Computer die Möglichkeit, statt des vollen Befehlswortes nur eine Abkürzung auf der Tastatur einzugeben. Meist wird aber dann das ganze Befehlswort in das Listing eingesetzt.

2. Weitere Befehle der voll ausgebauten BASIC-Dialekte

Es gibt heute kaum einen BASIC-Dialekt, der sich nur mit den Grundbefehlen der BASIC-Hochsprache begnügt. Auch wenn sie nicht ausdrücklich als »extended« bezeichnet werden, verarbeiten die meisten Dialekte noch viele weitere Befehle.

Hier können wir aber keine Liste aller bekanntgewordenen BASIC-Befehle aufstellen, denn dafür ist deren Anzahl inzwischen schon zu groß. Es werden darum nur Befehle besprochen, die in einer ganzen Reihe von BASIC-Dialekten vorkommen und die für allgemeine Programme wichtig sind.

2.1 Zusätzliche Funktionen

Einige BASIC-Versionen erlauben die Verwendung weiterer Funktionen. Eine Reihe davon, zum Beispiel die Hyperbel-Funktionen, werden nur für ganz bestimmte mathematische Aufgaben benötigt. Andere sind leicht abzuleiten. Die Bedeutung der meist verwendeten zusätzlichen Funktionen wird im folgenden erklärt.

BIN$ berechnet den Binärwert der als Argument eingesetzten Dezimalzahl.

CDBL
CINT
CSNG Bei den Computern, die das Rechnen mit verschiedener Genauigkeit zulassen, gibt es meist Funktionen, die eine Änderung des Variablentyps innerhalb des Programms bewirken. Bei Verwendung von CINT erfolgt manchmal auch kaufmännische Rundung.

DATE$ Das Datum wird als Zeichenkette zur Verfügung gestellt.

EOF Die Funktion meldet das Ende der sequentiellen Datei, deren logische Nummer als Argument eingesetzt ist.

FIX oder TRUNC Im Gegensatz zu INT schneidet diese Funktion die Nachkommastellen bei positiven und negativen Zahlen einfach ab.

HEX$ wandelt die im Argument stehende Zahl in eine Hexadezimalzahl um.

INSTR Mit dieser Funktion kann der Anfang eines Teilstrings innerhalb einer Zeichenkette, gegebenenfalls ab einem bestimmten Byte, ermittelt werden. Geliefert wird ein numerischer Ausdruck.

LOC Die Funktion meldet den zuletzt in einer wahlfrei lesbaren Datei bearbeiteten Datensatz.

LOF Die Funktion liefert die Anzahl der Datensätze in einer wahlfrei lesbaren Datei.

POS Die Funktion meldet bei einigen Dialekten die Cursor-Position in der Zeile.

STRING$ Als Argument sind bei dieser Funktion zwei Ganzzahlen einzusetzen. Die erste davon bestimmt, wie oft das Zeichen, dessen dezimaler ASCII-Wert als zweites im Argument steht, auf dem Bildschirm mit einem PRINT-Befehl ausgegeben werden soll.

TIME$ Die Uhrzeit wird in Form einer Zeichenkette zur weiteren Verarbeitung zur Verfügung gestellt.

2.2 Weitere Programmanweisungen

Anweisungen für den Programmlauf

BYE Das ursprünglich im Timesharing-Betrieb zum Abmelden verwendete Befehlswort wird jetzt manchmal für den Übergang von BASIC in den Monitor verwendet.

CLEAR Dieser Befehl löscht alle Variablenspeicher. Bei einigen Versionen bedeutet eine dem Befehl folgende Zahl die Anzahl der Speicherplätze, die für die Speicherung der Zeichen in Zeichenketten reserviert werden. Es gibt auch Versionen, bei denen mit diesem Befehl ein Speicherbereich abgetrennt werden kann, auf den BASIC dann keinen Zugriff hat.

CLS oder HOME Es gibt meist einen Befehl, mit dem die Anzeige auf dem Bildschirm gelöscht und der Cursor in die linke obere Ecke gesetzt werden kann. Manchmal muß allerdings hierfür auch ein Unterprogramm angerufen werden (**CALL CLEAR**).

DEF FN Mit diesem Befehl kann der Anwender selbst eine Funktion definieren. Die Syntax ist aber unterschiedlich, da bei einigen Dialekten nur numerische Funktionen, bei anderen auch Zeichenketten definiert werden können.

DEFDBL DEFINT DEFSNG DEFSTR Diese Befehle dienen zur Festlegung bestimmter Variablen-Namen auf bestimmte Variablentypen. Nach der Definition brauchen im weiteren Programm die Typenkennzeichen (#, %, !, $) nicht mehr verwendet zu werden.

DIV Operator für eine Ganzzahldivision.

GET oder INKEY$ Normalerweise nutzen wir bei BASIC den INPUT-Befehl für die Eingabe. Bei diesem Befehl besteht aber der Nachteil, daß einige Satzzeichen, wie das Komma, der Doppelpunkt und die Anführungszeichen, nicht mit eingegeben werden können. Bei sehr vielen BASIC-Versionen steht darum noch ein weiterer Eingabebefehl zur Verfügung, der jedes beliebige Zeichen von der Tastatur annimmt. Hierfür werden verschiedene Befehlswörter verwendet, und die Wirkung, auch die Syntax dieses Befehls variiert.

INP OUT Bei den Mikroprozessoren der Typen 8085 und Z 80 können 256 Ausgabekanäle direkt angesprochen werden. Computer, die einen dieser Prozessoren verwenden, haben meist auch Befehle, mit denen diese Kanäle in BASIC bedient werden können.

KEY zur Programmierung selbst zu programmierender Funktionstasten.

LINE INPUT Dieser verbesserte INPUT-Befehl erlaubt auch die Eingabe von Kommas, Doppelpunkten usw. und übernimmt alle Zeichen bis zum Wagenrücklauf-Kommando.

LOCATE Dieser Befehl erlaubt es, den Cursor auf eine bestimmte Bildschirmposition zu setzen. Beim nächsten PRINT-Befehl erscheint dann die Ausgabe an der

Stelle, deren Zeilen- und Spaltennummer im Befehl steht.

MAT Einige mathematisch orientierte BASIC-Dialekte enthalten besondere Befehle zur Matrizenverarbeitung.

MOD Dieser Operator dient in einigen Dialekten zur Bestimmung eines Divisionsrestes.

OPTION BASE setzt manchmal den ersten verwendbaren Index eines Feldes wahlweise auf 0 oder 1.

PRINT AT oder PRINT@ Diese Befehle erlauben die Positionierung der Ausgabe. Bei PRINT AT ist meist die Zeile und die Spalte, bei PRINT@ dagegen die (durchnumerierte) Bildschirmposition einzusetzen.

PRINT USING Dieser Befehl erlaubt die formatierte Ausgabe von Zahlen. Die Dezimalpunkte werden untereinander gesetzt und die Nachkommastellen nach Vorschrift gerundet. Die Anweisung, wie die Ausgabe erfolgen soll, muß meist in einer Zeichenkette vor dem Anruf des Befehls festgelegt werden.

RANDOMIZE Befehl, der in vielen Versionen dem Zufallsgenerator einen neuen Anfangswert zuweist.

REPEAT UNTIL Schleifenbefehle, bei denen der Abbruch erfolgt, wenn eine Bedingung am Ende nicht erfüllt ist. Manchmal auch als **DO ... UNTIL** geschrieben.

RESTOREz Im Gegensatz zur üblichen Wirkung des Befehls, der das Lesen wieder von der ersten DATA-Zeile ab veranlaßt, lassen es diverse Dialekte zu, daß nach dem Befehlswort diejenige Zeilennummer eingesetzt wird, von der an DATA-Zeilen gelesen werden sollen.

SWAP Der Befehl erlaubt das Tauschen der Namen zweier Variablen.

SCREEN Der Befehl ermöglicht die Anwahl verschiedener Bildschirmspeicher, manchmal auch die Wahl eines Grafikmodus.

WHILE Die Schleifenbefehle WHILE...END prüfen die Abbruchbedingung zu Beginn jedes Schleifenlaufs.

WAIT Computer, die mit dem Mikroprozessor 65xx arbeiten, haben oft einen Wartebefehl auch in BASIC. Die Regeln für die Verwendung sind aber unterschiedlich.

2.3 Anweisungen für Fehlerbehandlung im Programm

Findet der Interpreter in einem BASIC-Programm einen Fehler, sei es durch falsche Programmierung, sei es aber auch durch falsche Eingabe, so wird das Programm abgebrochen und der Fehler gemeldet. Dies kann sehr nachteilig sein, denn beim Neustart des Programms werden normalerweise die Inhalte der Variablenspeicher gelöscht.

Bei vielen BASIC-Versionen gibt es darum besondere Befehle, mit deren Hilfe der Programmabbruch beim Auffinden eines Fehlers vermieden werden kann. Im wesentlichen werden hierfür die folgenden Befehlswörter gebraucht:

ON ERROR GOTO Wird in das Programm an beliebiger Stelle dieser Befehl gesetzt, so erfolgt kein Programmabbruch, wenn der Computer irgendeinen Fehler feststellt. Der Programmlauf wird an dem im Befehl stehenden Sprungziel fortgesetzt. Dort muß ein Befehl stehen, der bestimmt, was bei einem Fehler geschehen soll.

ERL ERR In der Fehlerbehandlungsroutine kann untersucht werden, wo ein Fehler aufgetreten ist und um welchen Fehler es sich handelt. ERL gibt die Zeilennummer aus, in der der Fehler festgestellt wurde. ERR zeigt den vom Computer festgestellten Code des Fehlers.

RESUME Nach der Beendigung der Fehlerbehandlung kann mit diesem Befehlswort der normale Programmablauf fortgesetzt werden, wobei es für die Fortsetzung oft noch verschiedene Möglichkeiten gibt.

2.4 Ausgaben auf angeschlossene Drucker

Alle Home- und Personal-Computer bieten die Möglichkeit, Drucker anzuschließen und anzusprechen. Leider werden aber von den verschiedenen BASIC-Dialekten für die Bedienung externer Geräte recht unterschiedliche Methoden verwendet.

Anschluß über besondere Schnittstelle

Drucker werden heute fast allgemein entweder über die *serielle Schnittstelle V 24 (RS 232 C)* oder über die mit paralleler Datenübertragung arbeitende *Centronics-Schnittstelle* angesteuert. Es werden dann besondere Druckbefehle verwendet, an denen der Computer erkennt, daß er die Ausgabe nicht an den Bildschirm, sondern an den Drucker leiten soll. Üblich sind die folgenden Befehlswörter:

LLIST Die Programmliste wird auf dem Drucker erstellt. Der Ausdruck von Einzelzeilen oder Zeilenblöcken ist wie beim LIST-Befehl zu programmieren. Die gleiche Wirkung haben bei einigen Dialekten die Befehle **LIST/P** oder ähnliches.

LPRINT Der Befehl arbeitet genauso wie der einfache PRINT-Befehl. Komma, Semikolon, die TAB-Funktion und gegebenenfalls die Formatierung mit USING sind zulässig, nicht aber PRINT AT oder Befehle zur Cursorpositionierung. Die Ausgabe erfolgt nur auf dem Drucker.

Anschluß über IEEE-Bus

Einige weit verbreitete Computer (z. B. Commodore) haben keinen separaten Hardwareanschluß für den Drucker. Dieser muß an den für alle Peripheriegeräte zuständigen *IEEE-Bus* angeschlossen werden. Dies bedeutet, daß vor dem Ansprechen des Druckers immer erst per Software ein *Kanal* zum Drucker geöffnet werden muß. Hierfür sind folgende zwei Befehle üblich:

OPEN Der Öffnungsbefehl muß als Parameter meist noch einige Angaben wie Gerätenummer, logische Kanalnummer und Sekundäradresse enthalten. Nach der Öffnung des Kanals zum Drucker wirken die normalen PRINT-Befehle statt auf den Bildschirm nur auf den Drucker.

CMD Dieser Befehl, gleichfalls mit nachfolgenden Parametern, leitet auch die Ausgabe auf einen Ausgabekanal um. Er kann z. B. vor dem LIST-Kommando zur Ausgabe der Programmliste auf dem Drucker verwendet werden.

2.5 Programm- und Datenspeicherung auf Kassetten

Programmspeicherung

Die BASIC-Befehle für die Programmspeicherung sind zwar bei allen Computern ähnlich. In keinem Fall kann aber ein auf Kassette gespeichertes Programm in einen Computer einer anderen Bauart gelesen werden. Es gibt eine große Anzahl verschiedener Aufzeichnungsverfahren. Darüber hinaus sind aber auch noch die Methoden des Datenverkehrs zwischen Computer und Recorder unterschiedlich.

Meist besteht die Möglichkeit, bei der Kassettenspeicherung einen Programmnamen zu verwenden. Die Vorschriften für den Namen sind aber uneinheitlich. Einige BASIC-Dialekte unterstützen auch die Verwendung von 2 Recordern.

Die Kassettenrecorder werden immer über eine besondere Schnittstelle angeschlossen. Bei einigen Computern wird das Laufwerk durch den Computer gesteuert, bei anderen erfolgt eine Aufforderung hierzu auf dem Bildschirm. Vor- und zurückspulen muß allerdings immer der Benutzer selbst.

Bei den meisten BASIC-Versionen können angeschlossene Recorder mit besonderen Kommandos direkt angesprochen werden (Abb. 2). Üblich sind die Befehle:

LOAD oder CLOAD	Der Befehl zum Laden eines Programms kann als Parameter noch die Angabe des verwendeten Recorders (wenn mehr als einer angeschlossen werden kann) und eines Programmnamens erfordern.
SAVE oder CSAVE	Der Speicherbefehl für ein Programm ist meist ähnlich wie der Ladebefehl aufgebaut. Die Anfügung eines Programmnamens oder eines Kennbuchstabens ist meist zwingend.
CLOAD? oder VERIFY	Mit einem dieser Befehle kann geprüft werden, ob die Abspeicherung eines Programms einwandfrei geglückt ist. Das Programm bleibt gegebenenfalls für einen neuen Speicherversuch im Arbeitsspeicher stehen.

Vorgang	Apple	Atari	CBM VC-20	Genie TRS 80	MZ 80	Sinclair	TI 99/4 TI-BASIC
Programm laden	LOAD	CLOAD	LOAD"N"	CLOAD	LOAD"N"	LOAD"N"	OLD CS1
Programm speichern	SAVE	CSAVE	SAVE"N"	CSAVE	SAVE"N"	SAVE"N"	SAVE CS1
Speicherung prüfen	VERIFY	/	VERIFY	CLOAD?	VERIFY"N"	/	/
Öffnen für Datenspeicherung	/	OPEN #7, 4, 12, "N"	OPEN 1, 1, 0, "N"	/	WOPEN"N" ROPEN"N"	/	OPEN ....
Daten speichern	STORE*	PRINT #7	PRINT #1	PRINT #-1,	PRINT/T	/	PRINT #,
Daten lesen	RECALL*	INPUT #7	INPUT #1	INPUT #-1,	INPUT/T	/	INPUT #,
Trennzeichen	kein	CR	CR	CR, Komma	CR	/	CR, ; :
Öffnen zum Drucken	PRINT "PR #1"	/	OPEN 1, 4 : CMD 4	/	/	/	OPEN #255 ....
Programm drucken	LIST	LIST "P"	LIST	LLIST	LIST/P	LLIST	LIST #255
Daten drucken	PRINT	LPRINT	PRINT	LPRINT	PRINT/P	LPRINT	PRINT 255
Schließen Druckkanal	PRINT "PR #0"	/	CLOSE 1	/	/	/	CLOSE #255
	Auf Cassetten können nur eindimensionale numerische Felder gespeichert werden (*= Feldname)	Die Operanden im OPEN-Befehl müssen nach Vorschrift gewählt werden	Die Operanden im OPEN-Befehl müssen je nach Gerät und Vorgang gewählt werden	Angabe von Namen und Recorder-Nummer wahlweise möglich	Namen dienen zum Suchen auf dem Band	Nur Programmspeicherung, Datenspeicherung nicht möglich	Den OPEN-Befehlen folgen weitere Parameter

Abb. 2: Befehle für Kassettenbetrieb bei verschiedenen Computern

Datenspeicherung

Nicht alle BASIC-Versionen erlauben die Speicherung beliebiger Daten auf einer Kassette. Bei einigen Computern (z. B. Genie/TRS 80, Apple) werden für die Datenspeicherung auf Kassetten andere Befehle verwendet als bei Disketten. Es gibt aber auch Computer, die Kassettenrecorder und Diskettenstationen gleich behandeln.

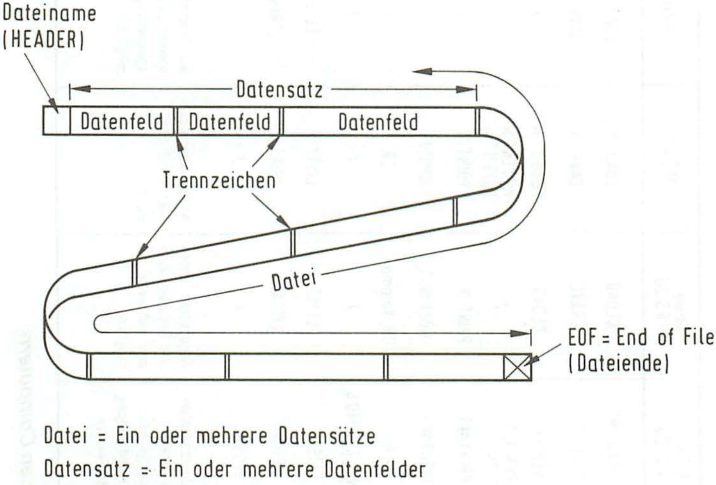


Abb. 3: Schema einer »sequentiellen Datei«

Wenn die Speicherung von Daten auf Kassetten erlaubt ist, erfolgt sie in Form der sequentiellen Datei (Abb. 3). Hierfür werden in den meisten Fällen die folgenden Anweisungen verwendet:

OPEN Für die Datenspeicherung muß meist mit diesem Befehl ein Pufferspeicherbereich und/oder ein Ausgabekanal bereitgestellt werden. Die dem Befehl folgenden Parameter stimmen nicht überein.

PRINT# Meist dient der PRINT#-Befehl mit nachfolgender Ziffer als Ausgabebefehl. Die weitere Syntax ist aber unterschiedlich. Oft können dem PRINT#-Befehl mehrere Konstante oder Variable folgen; sie müssen aber durch ein im betreffenden Dialekt zugelassenes Trennzeichen (Komma, Semikolon, CR o. a.) getrennt sein.

INPUT#	Der Datenübernahmebefehl entspricht weitgehend dem normalen INPUT-Befehl. Kommas sind als Trennzeichen zulässig.

Einige BASIC-Dialekte (z. B. Applesoft) erlauben die Datenspeicherung nur in Form numerischer Felder. Es werden dann folgende Befehle verwendet:

STORE ×	Das Feld mit dem Namen × wird komplett abgespeichert.
RECALL ×	Die gespeicherten Daten werden geladen und dem Feld × zugewiesen.

2.6 Befehle für Diskettenbetrieb

Für den Betrieb mit Disketten muß immer ein besonderes Betriebsprogramm tätig sein, das meist von einer Diskette zu laden ist. Es wird *Disketten Operations System* (DOS) genannt und unterstützt eine Reihe von DOS-Kommandos zur Verwaltung der Dateien auf einer Diskette und weitere Anweisungen, die im Programm für die Datenspeicherung und für das Wiedereinlesen der Daten in den Arbeitsspeicher verwendet werden können.

Grundsätzlich speichern und suchen alle DOS eine Datei nach dem vom Benutzer nach gewissen Regeln frei gewählten *Dateinamen*. Um den Ort, an dem die Daten physikalisch gespeichert sind, braucht er sich nicht zu kümmern.

Üblich sind auch direkte Kommandos in der Befehlsebene DOS, die oft durch ein besonderes Kommando anzurufen ist, wenn sonst in BASIC gearbeitet wird.

DIR oder CATALOG	zum Abrufen des Inhaltsverzeichnisses der auf einer Diskette gespeicherten Dateien.
SAVE	Speichern eines Programms auf eine Diskette.
LOAD	Laden eines Programms von einer Diskette.
FORMAT INIT NEW	Bevor eine Diskette benutzt werden kann, müssen in die einzelnen Spuren gewisse Kennwerte geschrieben werden. Hierfür sind verschiedene Kommandos gebräuchlich.

SCRATCH **KILL** **DELETE**	Auch zum Löschen einer auf der Diskette stehenden Datei werden verschiedene Befehlswörter verwendet.
COPY	Ein sehr wichtiger Befehl ist der Kopierbefehl, mit dem Dateien von einer Diskette auf eine andere übertragen werden können.

Den meisten Kommandos müssen noch einige Parameter folgen, deren Aufbau und Bedeutung bei den verschiedenen DOS sehr unterschiedlich ist.

Bei einigen DOS muß vor dem Ansprechen einer Diskette immer erst ein Datenkanal zur Diskette geöffnet werden. Dies ist notwendig, wenn Daten in eine Datei geschrieben oder daraus gelesen werden sollen. Hierfür wird stets der Befehl **OPEN** verwendet, bei dem allerdings die erforderlichen beziehungsweise zugelassenen Parameter sehr unterschiedlich aussehen. Im übrigen sind folgende Befehle für den Datenverkehr üblich:

PRINT# n	schreibt Daten in die unter Nummer n geöffnete Datei.
INPUT# n	liest Daten aus der Datei Nummer n.
WRITE	ähnlich PRINT#.
READ	ähnlich INPUT.
CLOSE	schließt geöffnete Dateien.
AP-PEND#n	fügt Daten an vorhandene Datei Nummer n an.

Alle DOS unterstützen *sequentielle Dateien*, bei denen sämtliche Daten nacheinander – nur durch bestimmte Trennzeichen getrennt – gespeichert werden. Zur Bearbeitung müssen sie als Ganzes in den Arbeitsspeicher geladen und danach auch als Ganzes wieder gespeichert werden.

Viele DOS erlauben die Anlage von *Dateien mit wahlfreiem Zugriff* auf einzelne Datensätze. Dies bedeutet, daß einzelne Daten verändert werden können, ohne daß die ganze Datei in den Arbeitsspeicher übernommen werden muß. Hier sind zwei verschiedene Typen üblich, die in Abb. 4 erklärt werden.

Bei Dateien mit fester Satzlänge muß diese in einem Parameter des Öffnungsbefehls festgelegt werden. In den Datensätzen können dann mehrere Variable (Datenfelder), durch Trennzeichen

getrennt, gespeichert werden. Die Datensätze werden gezählt und können unter ihrer Nummer aufgerufen werden. Oft wird hierfür der folgende Befehl verwendet:

RECORD a,b bewirkt, daß beim nächsten Lese- oder Schreibvorgang auf den Satz a, gegebenenfalls vom bten Byte an, zugegriffen wird.

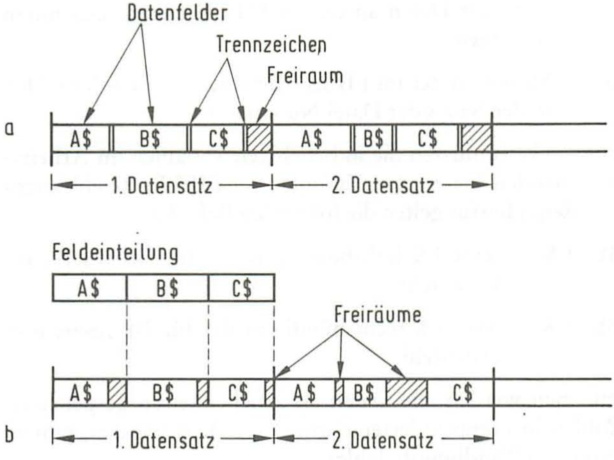


Abb. 4: Zwei Typen von wahlfrei lesbaren Dateien
a. Datei mit Trennzeichen zwischen den Datenfeldern
b. Datei mit fester Feldeinteilung in einem Befehl

Bei anderen Dialekten muß die Satz- und die Bytenummer in den Lese- oder Schreibbefehlen stehen.

Bei der zweiten gebräuchlichen Art wahlfrei lesbarer Dateien muß vorher der Aufbau der Datenfelder im Datensatz, d. h. die maximale Länge jeder zu speichernden Variablen, festgelegt werden. Für diese Feldeinteilung wird meist folgende Anweisung verwendet:

FIELD#n,B$ AS c,D$ AS e,F$ AS g. . .*

Im Befehl ist für jeden Variablennamen nach dem Befehlswort AS die Anzahl der Byte einzusetzen, die die Variable höchstens haben darf.

* Zur Bedeutung der Symbole in den Befehlen beachten Sie bitte den Kasten auf Seite 91.

Die Einteilung gilt für alle Sätze der unter Nummer n geöffneten Datei so lange, bis im Programmlauf eine neue FIELD-Anweisung angetroffen wird. Es ist also möglich, in einer Datei mit verschiedenen Einteilungen zu arbeiten.

In dem Programm gelten dann folgende Anweisungen:

GET n,s Lesen des Satzes s der Datei Nummer n und Zuweisung der Daten an die im FIELD-Befehl genannten Variablen.

PUT n,s Speichern der im FIELD-Befehl genannten Variablen in den Satz s der Datei Nummer n.

Vor dem Speichern müssen die in beliebigen Variablen im Arbeitsspeicher stehenden Daten den Variablen im FIELD-Befehl zugewiesen werden. Hierfür gelten die folgenden Befehle:

LSET B$=E$ setzt E$ linksbündig in das für B$ reservierte Datenfeld.

RSET B$=E$ setzt E$ rechtsbündig in das für B$ reservierte Datenfeld.

Im allgemeinen werden alle Daten als Zeichenketten gespeichert. Sollen Zahlen in komprimierter Form gespeichert werden, gibt es dafür besondere Wandlungsbefehle:

LSET B$=MKI$(E%) setzt die Integervariable E% in das 2-Byte-lange Feld B$.

LSET C$=MKS$(E!) setzt die Variable mit einfacher Genauigkeit E! in das 4-Byte-lange Feld C $.

LSET DS=MKD$(E#) setzt die doppelt genaue Variable E# in das 8-Byte-lange Feld D$.

Weitere vielseitig zu nutzende Dateitypen bieten die Betriebssysteme NEWDOS und G-DOS, bei denen die Vorteile der sequentiellen und der wahlfrei lesbaren Dateien vereinigt werden.

Von einigen DOS wird auch noch ein Dateityp angeboten, bei dem der Zugriff auf einzelne Daten möglich ist. Jedoch muß der Programmierer hier die Adressenverwaltung (Spur und Sektor auf der Diskette) selbst durchführen. Diese Dateien erfordern darum erheblichen Programmaufwand und gute Programmierkenntnisse.

Meist gibt es dann noch Befehle für sogenannte *Binärdateien*, mit denen ganze Speicherbereiche des Arbeitsspeichers als Datei auf eine Diskette geschrieben werden können.

2.7 Zugriff auf die Maschinensprache

Auch bei der Programmierung in BASIC arbeitet ein Computer intern immer nur mit den Befehlen der Maschinensprache, die der als Zentraleinheit verwendete Mikroprozessor versteht. Die BASIC-Befehle werden durch den Interpreter in die Maschinensprache übersetzt.

Es gibt aber in allen voll ausgebauten BASIC-Dialekten auch Befehle, mit denen innerhalb eines BASIC-Programms direkt auf die Speicheradressen des Arbeitsspeichers zugegriffen werden kann. Weiterhin ist es möglich, Unterprogramme in Maschinensprache in einem BASIC-Programm anzurufen.

Direkter Zugriff auf die Adressen des Arbeitsspeichers

PEEK Wenn in die Klammer dieser Funktion als Argument eine Speicheradresse geschrieben wird, stellt sie den dort gespeicherten Inhalt als Dezimalzahl von 0 bis 255 zur Verfügung. Wird die Funktion in einen PRINT-Befehl geschrieben, wird der Wert ausgegeben.

Obwohl bei allen mit einem Mikroprozessor arbeitenden Computern Adressen von 0 bis 65536 ansprechbar sind, können manchmal nur die Adreßinhalte des Schreib-Lese-Speichers (RAM) geholt werden. Ist eine Adresse hardwaremäßig nicht besetzt, kann es eine Fehlermeldung geben.

Die Adresse muß meist als Dezimalzahl in die Klammer der Funktion gesetzt werden. Hierbei gibt es aber noch zwei Dialektvarianten. Bei neueren Dialekten ist es oft möglich, dort Dezimalzahlen von 0–65535 einzusetzen, andere Dialekte speichern die Zeilennummern als Ganzzahl von −32768 bis +32767. Alle Zeilennummern über 32767 müssen dann in die Komplementzahl umgerechnet werden (siehe das Beispiel in Absatz 7.5).

POKE Mit diesem Befehl werden bestimmte Bitmuster in die im Befehl stehende Adresse geschrieben. Adresse und Bitmuster sind als Dezimalzahlen durch ein Komma getrennt einzusetzen.

VARPTR Da BASIC mit dynamischer Speicherverwaltung arbeitet und der Programmierer keinen Einfluß auf die Adressen hat, in denen Variable abgelegt werden, gibt es bei neueren BASIC-Dialekten oft eine Funktion, mit der die Adressen abgefragt werden können. Für ihre Anwendung muß man die Art, wie der betreffende Computer die Variable speichert, genau kennen (siehe Beispiel in Abb. 11, Seite 86).

Die für den Speicherzugriff verwendeten Befehle sind zwar in allen BASIC-Dialekten gleich. Trotzdem ist es sehr schwer, Programme, in denen direkt auf Speicheradressen zugegriffen wird, zum Lauf auf einem anderen Computer umzuschreiben. Auch wenn der gleiche Mikroprozessor in zwei verschiedenen Computern verwendet wird, heißt dies noch lange nicht, daß für gleiche Zwecke auch gleiche Adressen benutzt werden. Eine Programmkonvertierung setzt also eine genaue Kenntnis der Speicherbelegung beider Computer voraus.

Inwieweit überhaupt direkte Zugriffe auf Speicheradressen für die Programmierung notwendig sind, hängt im wesentlichen vom Ausbau der verwendeten BASIC-Version ab. Gut und benutzerfreundlich ausgestattete Versionen haben für alle denkbaren Aufgaben BASIC-Befehle. Im Gegensatz dazu erfordern ältere und weniger komfortable Dialekte vielfach direkten Speicherzugriff mit POKE-Befehlen, wenn im Programm bestimmte Vorgänge ablaufen sollen. So kann man zum Beispiel die Negativdarstellung bei vielen Dialekten durch einen BASIC-Befehl (INVERSE) veranlassen, während man bei anderen mit POKE in eine bestimmte Speicherstelle einen bestimmten Wert setzen muß.

Verwendung von Unterprogrammen in Maschinensprache

Es gibt fast immer eine Möglichkeit, in einem BASIC-Programm auch Routinen der Maschinensprache aufzurufen. Die Befehle hierfür sind in allen Dialekten ähnlich, aber in ihrer Wirkung doch recht uneinheitlich.

CALL Oft kann ein Unterprogramm einfach mit dem CALL-Befehl plus nachfolgendem Programmnamen aufgerufen werden. Dies gilt sowohl für fest in den Interpreter eingebaute Programme, als auch für Maschinensprache-Unterprogramme, die der Benutzer selbst schreibt.

USR oder SYS In den meisten Fällen ruft man vom Benutzer geschriebene Unterprogramme mit der USR-Funktion an, die im Argument einen Hinweis auf die Speicheradresse des Programms enthalten muß. Hier sind verschiedene Methoden üblich. Parameterübergabe kann vorgesehen sein.

DEFUSR Die Speicheradressen für mehrere Unterprogramme können oft zu Beginn des Programms definiert werden.

Wenn selbstgeschriebene Maschinensprache-Unterprogramme verwendet werden sollen, ist es meist notwendig, dafür Speicherraum freizuhalten, auf den BASIC nicht zugreifen kann. Oft erfolgt dies bereits beim Einschalten auf die Frage MEM SIZE?. Auch der CLEAR-Befehl wird hierfür mitverwendet.

Hilfen für Programmierung in Maschinensprache

In Maschinensprache geschriebene Programmteile können durch eine Schleife mit dem POKE-Befehl direkt in Speicheradressen geschrieben werden. Sind die Befehle hexadezimal codiert, müssen sie vorher in Dezimalzahlen gewandelt werden. Einige BASIC-Dialekte bieten aber auch die Möglichkeit, Hexadezimalzahlen in BASIC-Programmen zu verwenden.

Oft gibt es ein besonderes Programm, MONITOR genannt, das die Programmierung in Maschinensprache erleichtert. Anruf und Aufbau dieser Programme fallen aber recht verschieden aus.

2.8 Kommandos zur Erleichterung der Programm-Entwicklung

Viele BASIC-Versionen haben Kommandos (die zum Teil auch als Anweisungen laufen), mit denen das Programmieren erleichtert wird. Die wichtigsten Befehle aus dieser Gruppe lauten wie folgt:

AUTO Nach diesem Kommando wird nach jedem RETURN selbsttätig eine neue Zeilennummer an den Anfang der Zeile gesetzt, wobei die Anfangszeilennummer und die Schrittweite gewählt werden können.

EDIT Bei einigen BASIC-Versionen ist für die Korrektur einer schon gespeicherten Zeile der Anruf eines

besonderen Betriebsprogramms mit dem Namen EDITOR notwendig. Manchmal geht der Computer auch automatisch in die Betriebsart EDITOR, wenn im Programmlauf ein Fehler gefunden wurde.

TRACE NOTRACE	Einige Computer erlauben einen TRACE-Betrieb, bei dem der Computer die Zeilennummern aller verarbeiteten Zeilen auf den Bildschirm schreibt, so daß man verfolgen kann, wie das Programm gelaufen ist. Es werden statt TRACE auch **TRON** und **FLOW** und statt NOTRACE auch **TROFF, NOFLOW** und **UNTRACE** verwendet.
STEP	Manche BASIC-Versionen ermöglichen den Schritt-Betrieb, bei dem das Programm jeweils nur einen Befehl ausführt und dann wartet, bis die RETURN-Taste betätigt wurde.
RENUM	Dieses Kommando erlaubt bei vielen Dialekten die Neunumerierung eines im Arbeitsspeicher stehenden Programms. Komfortable Dialekte gestatten es zusätzlich, nur bestimmte Teile eines Programms zu ändern. Dann kann die Struktur mit Hunderternummern am Anfang eines Programm-Moduls erhalten bleiben. Bei einfacheren Dialekten können nur alle Befehle von einer bestimmten Zeilennummer ab neu numeriert werden. Gute BASIC-Versionen rechnen bei diesem Kommando alle Sprungziele selbsttätig um.
DELETE	Mit diesem Kommando kann eine oder können mehrere Zeilen im Programm gelöscht werden.

3. Grafik-, Farb- und Tonbefehle

Wie alle älteren Programmiersprachen, so war auch BASIC ursprünglich nur für Datenverarbeitung konzipiert und enthielt darum auch keine besonderen Grafikbefehle oder Befehle für die Tonerzeugung. Mit dem Aufkommen der Personal- und Homecomputer wurde die Sprache kontinuierlich erweitert, wobei leider jeder Computerhersteller eigene Wege ging. So gibt es heute eine fast unübersehbare Menge von Sonderbefehlen, die in BASIC-Programmen verwendet werden können, um spezielle Effekte zu erzielen.

Es ist unmöglich, an dieser Stelle alle Dialektvarianten, die bei den verschiedenen Computern vorkommen, im einzelnen zu behandeln. Dieses Kapitel beschränkt sich darum auf die Erklärung der im wesentlichen angewendeten Methoden. Grundsätzlich muß man dabei zwei Arten der Befehlsgebung unterscheiden.

Da der Platzbedarf im Arbeitsspeicher heute keine große Rolle mehr spielt, können die Interpreter moderner BASIC-Versionen viel mehr Befehle verarbeiten als früher. Man hat sich darum bemüht, auch für die zusätzlichen Aufgaben der Grafikdarstellung und Tonerzeugung die Syntax der Befehle so zu gestalten, daß sie für den Benutzer möglichst einfach zu handhaben ist. Dies trifft auf ältere BASIC-Dialekte nicht zu. Durch zusätzlich eingebaute Hardware-Bauelemente mögliche Computerfunktionen können hier nicht durch BASIC-Befehle, sondern nur durch Eingriffe in die Maschinensprache programmiert werden. Beide Methoden werden hier behandelt.

3.1 PRINT- und POKE-Grafik

Auch die einfachsten BASIC-Dialekte gestatten es, die Zeichen, die in ihrem Zeichenvorrat vorhanden sind, mit dem PRINT-Befehl an beliebige Stellen des Bildschirms zu setzen. Entweder gibt es hierfür einen Befehl zur Cursorpositionierung, oder es besteht die Möglichkeit, die Tastenkommandos zur Cursorpositionierung mit in das Programm einzubauen (siehe auch Abschnitt 4.6). Die erste Methode ist einfacher zu handhaben. Nur schwer zu verstehen sind oft Programme, die mit einem Computer geschrieben wurden, der nach der zweiten Methode arbeitet.

Da die druckbaren Zeichen des ASCII-Code nur die Codewerte mit Dezimalwerten von 32 bis 127 umfassen, in einem Byte aber 256 verschiedene Bitmuster zur Verfügung stehen, haben viele Computerhersteller den ASCII-Code ergänzt oder arbeiten sogar mit einem besonderen Bildschirmcode, in dem alle 256 Möglichkeiten ausgenutzt werden.

Wenn nur die Dezimalwerte von 128 bis 255 mit zusätzlichen Symbolen belegt werden, bleibt der ASCII-Code bestehen, und es ergeben sich bei normalen Programmen keine Schwierigkeiten bei der Konvertierung. Wenn der Bildschirmcode aber wesentlich vom ASCII-Code abweicht, kann man die Programme nur verstehen, wenn man ersteren genau kennt.

Es besteht oft die Möglichkeit, mit dem POKE-Befehl die Zeichen direkt in beliebige Speicherstellen des Bildschirmspeichers zu schreiben. Die Verwendung der PRINT-Grafik ist aber einfacher.

Sowohl bei der PRINT- als auch bei der POKE-Grafik wird in jede Bildschirmposition ein Zeichen gesetzt. Die Anzahl der zu setzenden Zeichen ist also nicht größer als im Text-Modus. Um trotzdem zu einer größeren Auflösung, also zu einer größeren Anzahl darstellbarer Punkte zu kommen, bestehen die Grafikzeichen aus 4 bis 8 Einzelfeldern, die auch *Pixel* genannt werden. Wenn der Bildschirm normalerweise die Darstellung von 1000 Zeichen zuläßt, können auf diese Weise 4000 bis 8000 Einzelpunkte gezeigt werden.

Wird mit Pixeln gearbeitet, werden diese entweder so groß ausgelegt, daß keine Zwischenräume zwischen den Zeilen bleiben, oder es wird ein besonderer Grafik-Modus verwendet, bei dem die Zeilen enger zusammenrücken. In jedem Fall ist es bei der Verwendung von Pixeln möglich, Linien und Flächen auf dem Bildschirm ohne Zwischenräume darzustellen.

3.2 Spezielle Grafikbefehle

Die Problematik einer grafischen Darstellung auf dem Bildschirm liegt in der Schwierigkeit, eine genügend hohe Auflösung in Einzelpunkte mit einem noch akzeptablen Speicherbedarf in Einklang zu bringen. Homecomputer haben Bildschirmspeicher mit ca. 500 bis 1024 Byte Kapazität, während Personalcomputer dafür ca. 2000 Byte zur Verfügung stellen.

Wenn nur jede Bildschirmposition als Punkt betrachtet wird, ergibt sich eine völlig ungenügende Auflösung. Mit Hilfe der Pixel-Technik kann zwar die Auflösung auf das Vier-, Sechs- oder Achtfache verbessert werden; aber das Einschreiben der Codewerte für die Pixel mit Hilfe von POKE-Befehlen ist doch recht mühsam. Viele BASIC-Dialekte bieten darum für höher aufgelöste Grafik besondere Befehle.

Dabei kann sich die Speicherorganisation sehr unterschiedlich darstellen. Einige Dialekte kommen auch bei höherer Auflösung mit dem normalen Bildschirmspeicher aus. Bei anderen wird im Grafikbetrieb mehr Speicherraum benötigt, der dann durch einen besonderen Befehl vor der Anwendung von Grafikbefehlen reserviert werden muß. Es gibt Computer, die dabei bis zu 12 verschiedene Grafik-Modi zulassen, die sich bezüglich der Auflösung, der Farbmöglichkeiten und der Aufteilung des Bildschirms unterscheiden.

Am häufigsten werden die folgenden Grafikbefehle verwendet:

GRAPHICS oder **GR** — Aufrufbefehl für den Grafikbetrieb. Oft sind Parameter notwendig. Es sind jedoch auch völlig andere Befehlswörter wie **SCREEN, PMODE, PCLEAR** üblich. Manchmal können durch diesen Befehl mehrere Bildschirmspeicher reserviert werden, auf die wiederum im Programm wahlfrei zugegriffen werden kann.

SET oder **PSET** oder **PLOT** — Mit diesem Befehl, dem die x- und y-Koordinaten eines Bildschirmpunktes folgen müssen, wird oft ein bestimmter Punkt gesetzt. Vielfach wird die linke obere Ecke des Bildschirms als Koordinatennullpunkt angenommen. Es gibt aber auch Dialekte, die mit einem anderen Nullpunkt arbeiten.

DRAW — Dieser Befehl dient vielfach zum Zeichnen einer Linie. Es müssen dann die Koordinaten der Anfangs- und Endpunkte eingesetzt werden. Manchmal wer-

den hierfür die Befehlswörter **LINE** oder **PLOT** verwendet.

DRAWTO Wenn der Zeichenbefehl in dieser Form zu geben ist, folgt meist nur die Koordinate für den Endpunkt der zu ziehenden Linie.

RESET löscht einen vorher gezeichneten Punkt. **PRESET** und andere Befehlswörter sind gleichfalls üblich.

POINT testet, ob ein Punkt, dessen Koordinaten im Befehl stehen, gesetzt ist. Andere Befehlswörter kommen ebenfalls vor.

HLINE zieht eine waagerechte Linie.

VLINE zieht eine senkrechte Linie.

CIRCLE zeichnet einen Kreis. Bei einigen Versionen ist zusätzlich das Zeichnen einer Ellipse oder eines Kreisabschnitts möglich.

LINE Durch einen Zusatz kann der Befehl zum Zeichnen einer Linie manchmal zum Zeichnen eines Rechtecks verwendet werden.

3.3 Programmierung von besonderen Bildzeichen (Sprites)

Jedes auf den Bildschirm geschriebene Zeichen wird einem Zeichenspeicher (Zeichengenerator) im ROM entnommen, wo es meist in Form einer Matrix mit 8 Zeilen und 8 Spalten gespeichert ist. Für die Darstellung auf dem Bildschirm kann in jede Matrixposition ein Punkt gesetzt werden. Bei Einzelansteuerung der Punkte würde sich also eine recht hohe Auflösung für das Bild auf dem Schirm ergeben.

Die meisten BASIC-Versionen erlauben aber keine direkte Einzelansteuerung der Bildschirmpunkte. Dafür bieten BASIC-Versionen, die auch für die Bilddarstellung von Spielvorgängen ausgelegt sind, die Möglichkeit, besondere Bildzeichen als Matrix zu programmieren und diese dann auf den Bildschirm zu bringen. Man nennt diese Zeichen auch *Sprites*. Bei benutzerfreundlichen Dialekten gibt es hierfür besondere Befehle. Bei weniger ausgearbeiteten Dialekten müssen POKE-Befehle verwendet werden.

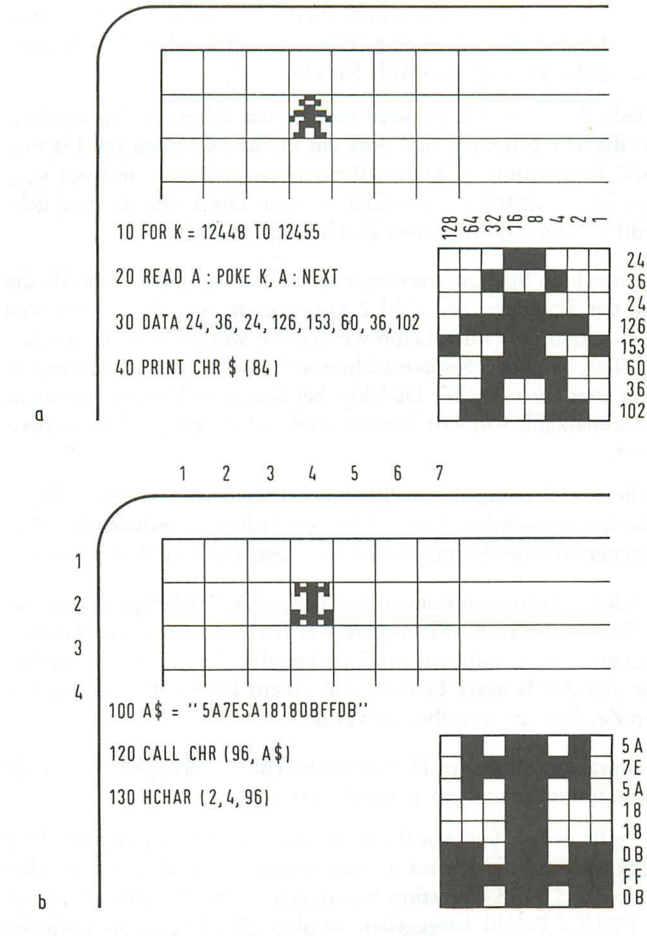


Abb. 5: Zwei Methoden für die Programmierung von Bildzeichen

Die auf dem Bildschirm dargestellten Zeichen werden intern im Computer als 8×8-Matrix gespeichert. Es gibt oft Methoden, eigene Zeichen zu programmieren, die allerdings bei den einzelnen Computern sehr unterschiedlich sind. Hier zwei Beispiele:

a. Die Belegung der einzelnen Zeilen muß als Dezimalzahl in bestimmte Speicherstellen des Zeichenspeichers geschrieben werden. Die Darstellung erfolgt durch PRINT- oder POKE-Befehle.

b. Die Belegung der einzelnen Zeilen ist als Hexadezimalzahl in eine Zeichenkette zu schreiben. Diese wird mit einem Unterprogramm in einen bestimmten Speicherbereich geschrieben und kann von dort abgerufen werden

Üblich ist es, jede Zeile der Zeichenmatrix in einem Byte darzustellen, so daß für jedes Zeichen 8 Byte benötigt werden. Das Schema für diese Darstellung ist in Abb. 5 erklärt.

Für jede Zeile der Matrix wird eine Binärzahl gespeichert, wobei meist die »1« bedeutet, daß dort ein Punkt zu setzen ist. Da nun BASIC-Programme nicht mit Binärzahlen arbeiten, müssen statt dessen Dezimalzahlen verwendet werden. Die Umkodierung geht am einfachsten mit Hilfe einer Zeichnung der Matrix.

Den einzelnen Spalten werden, von rechts beginnend, jeweils die Werte der Potenzen der Zahl 2 zugewiesen, wie dies im binären Zahlensystem üblich ist. Dann werden die Werte der in den einzelnen Zeilen besetzten Spalten addiert und als Zeilenwert verwendet. Es gibt aber auch BASIC-Dialekte, bei denen die Zeilenwerte nicht als Dezimalzahl, sondern hexadezimal codiert eingegeben werden müssen.

Für die Speicherung der so definierten Zeichen werden mehrere Methoden verwendet. Einige Dialekte halten innerhalb des Zeichengenerators bestimmte Plätze für selbstdefinierte Zeichen frei.

Bei anderen Versionen muß vorher der ganze Zeichengenerator aus dem ROM in das RAM kopiert werden. Um auf diese Zeichen zurückgreifen zu können, muß vorher der Zeiger, der normalerweise den Zeichensatz bedient, mit einem POKE-Befehl auf den neuen Zeichensatz umgebogen werden.

Manchmal kann man im Definitionsbefehl für das Sprite eine Zahl wählen, unter der es dann abzurufen ist.

Sonst gibt es für den Abruf der definierten Zeichen verschiedene Methoden. Sind die Sprites im Zeichengenerator abgelegt, werden sie mit der CHR$-Funktion bereitgestellt und können dann mit dem PRINT-Befehl ausgegeben werden. Als Argument muß die Codezahl des zum Speicherplatz gehörigen Bildschirmcode in die Funktion eingesetzt werden.

Auch mit einem POKE-Befehl, der nach dem Komma die Codezahl enthalten muß, können Sprites auf den Bildschirm gesetzt werden. Bewegung wird durch Einschreiben eines Sprites in eine Bildschirmposition und durch Einschreiben eines Leerstrings in die vorher besetzte Position dargestellt.

Neuere BASIC-Dialekte, die besondere Befehle zum Setzen der Sprites besitzen, sind einfacher zu programmieren. Man kann dort die Position, in der das Zeichen erscheinen soll, als Koordinaten

einsetzen und dann schrittweise Änderungen zur Darstellung von Bewegungen programmieren.

Es gibt BASIC-Versionen, die es erlauben, größere Sprites (zum Beispiel in einer 16-mal-16-Punkte-Matrix oder noch größer) zu definieren. Manchmal ist es auch möglich, eine Anzahl von selbst definierten Zeichen zu einem größeren Objekt zusammenzufügen und dieses dann als Ganzes zu bewegen.

Diese kurzen Ausführungen zeigen schon, daß es sehr viele Varianten für die Darstellung besonderer Zeichen in den verschiedenen BASIC-Dialekten gibt. Es ist darum außerordentlich schwer, ein mit bestimmter Technik arbeitendes Programm auf eine andere BASIC-Version umzustellen.

3.4 Programmierung von Figuren (Shapes)

In der Grafik gut ausgebaute BASIC-Versionen bieten noch eine weitere Möglichkeit, größere *Figuren* zu programmieren und dann nur mit einem Befehl aufzurufen. Manchmal wird von der Einrichtung einer SHAPE TABLE gesprochen, aber auch andere Ausdrücke finden Verwendung.

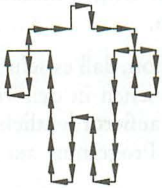
Das Prinzip besteht darin, die Koordinaten für mehrere Zeichenbefehle zusammenzufassen, damit sie als Ganzes aufgerufen und zur Wirkung gebracht werden können. In den einzelnen BASIC-Dialekten werden verschiedene Programmiermethoden verwendet. Zwei davon sollen hier erklärt werden.

Es gibt Dialekte, bei denen in den Zeichenbefehl DRAW ein String mit mehreren hintereinander stehenden Zeichenanweisungen gesetzt werden kann. Unabhängig vom DRAW-Befehl können dann die Zeichenanweisungen auch einer String-Variablen zugewiesen werden, so daß die ganze Figur schnell abrufbar ist. Wie das nach diesem Verfahren arbeitende BASIC des DRAGON-Computers arbeitet, zeigt Abb. 6b.

Komplizierter wird die Programmierung, wenn sie auch noch die Möglichkeit bietet, eine Figur in der Größe zu ändern oder rotieren zu lassen. Man muß dann die Zeichenanweisungen in einer Figuren-Tabelle festlegen. Hier werden wieder die verschiedensten Verfahren angewandt.

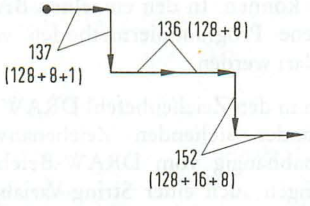
Beim Colour-Genie werden zum Beispiel jeweils die Zeichenanweisungen für zwei Zeichenschritte als Vektoren zusammengefaßt und

Abb. 6: Darstellung von Figuren (Shapes)
Für den Aufbau von Figurentafeln gibt es in den verschiedenen BASIC-Dialekten unterschiedliche Verfahren. Die Figur wird zwar immer durch hintereinander geschaltete Vektoren gezeichnet; aber manchmal ist die Programmierung etwas kompliziert.



a. Beim Colour Genie, zum Beispiel, werden die Daten für zwei Vektoren in einem Byte gespeichert. Nach einer Tabelle sind einzelne Bit zu setzen; dann ist das Bitmuster als Dezimalzahl einzugeben.

	128	64	32	16	8	4	2	1	
	1. Vektor				2. Vektor				
→	1	0	0	0	1	0	0	0	→
↓	1	0	0	1	1	0	0	1	↓
←	1	0	1	0	1	0	1	0	←
↑	1	0	1	1	1	0	1	1	↑
	Farbe		Richtung		Farbe		Richtung		



U = Bewegung nach oben
D = Bewegung nach unten
L = Bewegung nach links
R = Bewegung nach rechts
E = diagonal nach oben rechts
F = diagonal nach unten rechts
G = diagonal nach unten links
H = diagonal nach oben links

b. Bei neueren Microsoft-BASIC-Versionen wird ein Befehlsstring gebildet, in den verschiedene Vektoren nacheinander mit jeweils einem Kennbuchstaben und der Längenangabe eingesetzt werden

```
A$ = "R1D1R2D1R1D2L1U2L1 ..........
DRAW A$
```

mit den Informationen für die Farbgebung in einem Byte abgelegt (Abb. 6a). Sie müssen mit dem POKE-Befehl in bestimmte Speicherstellen geschrieben werden. Vorher ist auch noch die Anzahl der Byte, die für eine Figur benötigt werden, entsprechend abzuspeichern.

Die Länge der Vektoren wird vor dem Zeichnen der ganzen Figur durch den Befehl **SCALE n** festgelegt; n bedeutet dabei die Anzahl der kleinstmöglichen Zeichenschritte, die in der gewählten Grafikdarstellung möglich ist.

Zeichnen und Löschen erfolgt mit Hilfe des SHAPE- beziehungsweise NSHAPE-Befehls, in den die Koordinaten des Punktes einzusetzen sind, an dem das Zeichnen der Figur beginnen soll.

3.5 Befehle für die Farbgebung

Obwohl vielfach die gleichen Befehlswörter verwendet werden, sind die Befehle, die bei den verschiedenen BASIC-Dialekten für die Farbausstattung bei Anschluß eines Farbfernsehers oder eines Farbmonitors gegeben werden können, in der Syntax sehr verschieden. Es können darum hier nur einige Grundlagen erklärt und einige Beispiele vorgestellt werden.

Grundsätzlich unterscheiden wir zwischen der *Hintergrundfarbe*, in der der Bildschirm erstrahlt, wenn dort nichts dargestellt ist, und der *Vordergrundfarbe*, in der eine Darstellung von Zeichen oder Figuren auf dem Hintergrund erscheint. Manchmal gibt es dann noch eine besondere Farbe für den Bildrand, die auch *Borderfarbe* genannt wird.

In der Anzahl der Farben und Farbtöne weichen die einzelnen Computer stark voneinander ab. Je mehr Farben ansteuerbar sind, desto mehr Speicherraum wird benötigt. Es ist darum üblich, bei höherer Grafikauflösung (die auch mehr Speicherraum erfordert) die Anzahl der Farben zu beschränken.

Oft wird zur Speicherung der Farbinformation für die einzelnen Bildpunkte ein besonderer *Farbspeicher* unabhängig vom normalen *Bildschirmspeicher* verwendet, der für jede Bildschirmposition die Farbinformation aufnimmt. Bei einigen Dialekten muß man diesen Farbspeicher allerdings mit POKE-Befehlen programmieren.

Bei höherer Auflösung wird der Farbspeicher manchmal mit dem Zeichenspeicher für die einzelnen darstellbaren Punkte verbunden.

Es gibt aber auch Dialekte, die mit *Farbregistern* arbeiten. In ihnen werden nur die Farbinformationen abgelegt, die dann jeweils für alle nachfolgenden Befehle gelten, bis eine neue Farbinformation erteilt wird.

Vielfach werden die folgenden Befehlswörter verwendet, deren Argumente aber differieren können oder unterschiedliche Bedeutung haben.

COLOR oder COLOUR	Der Grundbefehl für die Vordergrundfarbe gilt vielfach nur im sogenannten Textmodus, also bei nicht erhöhter Bildschirmauflösung, wenn mit dem PRINT-Befehl gearbeitet wird. Die Farben können ebenso im Direktbetrieb durch Betätigung spezieller Tasten oder Tastenkombinationen aufgerufen werden. Manchmal werden mit diesem Befehl auch Vorder- und Hintergrundfarbe definiert.
SCREEN	Mit diesem Befehl werden gelegentlich Vordergrund- und Hintergrundfarbe sowie die Darstellungsart auf dem Bildschirm gewählt.
PAINT	Dieser Befehl dient zum Ausfüllen einer schon gezeichneten Kontur mit einer bestimmten Farbe.

3.6 Tonerzeugung auf Homecomputern

Als die Programmiersprache BASIC entwickelt wurde, hat man mit Sicherheit nicht daran gedacht, daß ein Computer echte Musik machen kann. Zwar genügen die BASIC-Grundbefehle, um jedem Computer einfache Töne und Geräusche zu entlocken, aber angenehm klingende Musik kann ein Computer nur ertönen lassen, wenn er entsprechende Hardware-Bauelemente besitzt.

Einfache Töne werden erzeugt, indem die Stromzufuhr zu einem Lautsprecher in einem bestimmten Rhythmus geschaltet und wieder unterbrochen wird. Dies kann ohne weiteres in BASIC-Schleifen geschehen. Wird der Strom in einer Sekunde 440mal ein- und ausgeschaltet, entsteht der Kammerton a.

In viele Computer ist ein kleiner Lautsprecher eingebaut, der dann durch BASIC-Befehle angesprochen werden kann. Üblich sind:

BEEP läßt einen kurzen Ton erklingen, der im Programm

auch als Warnsignal oder für sonstige Zwecke verwendet werden kann.

KLIK Computer, die keine beweglichen Tasten besitzen, geben manchmal ein kurzes Tonsignal aus, wenn eine Tastenbetätigung vom Computer angenommen wurde.

Wesentlich mehr Möglichkeiten für die Tonerzeugung eröffnen sich, wenn auch die Tonhöhe und die Tondauer programmiert werden können. Verwendet wird hierfür gelegentlich das Befehlswort **BEEP**; üblich ist aber **SOUND.** Manchmal ist auch ein Unterprogramm zur Tonerzeugung anzurufen. Leider sind die Parameter, die dem SOUND-Befehl folgen müssen, in den einzelnen BASIC-Dialekten sehr unterschiedlich angelegt.

Zur Bestimmung der Tonhöhe sind zwei Methoden üblich. Einige Dialekte verlangen die Programmierung der Schwingungsdauer zum Beispiel in Hertz (Schwingungen pro Sekunde), während sich andere am System der Notenschreibung orientieren (Abb. 7).

Muß zum Programmieren bestimmter Töne die Schwingungszahl in den Befehl geschrieben werden, kann man diese meist vorher entsprechend der gewünschten Tonart als Variable definieren. Es ist auch üblich, für jeden Ton ein Unterprogramm anzurufen.

Für das zweite Verfahren muß ein der Note entsprechender Code in den Befehl geschrieben werden. Bei diesem Code gibt es aber sehr viele Variationen. Meist wird zwischen den Codewerten für die Töne einer Oktave und einem besonderen Wert für die Wahl der Oktave unterschieden. Oft verwendet man für beides Zahlen. Ebenfalls üblich sind Buchstaben, mit denen die Noten bezeichnet werden, und gewisse Grafiksymbole.

Die Tondauer wird immer durch eine Zahl bestimmt. Manchmal bedeutet diese Sekunden, manchmal aber Millisekunden. Auch Codierungen sind üblich. Es dürfen dann Zahlen von 1 bis 255 eingesetzt werden, und 16 bedeutet zum Beispiel eine Sekunde. In einigen Dialekten läßt sich die Tondauer allerdings nur in Form einer Schleife programmieren.

Noch besser wird die Tonqualität, wenn auch die Lautstärke mitprogrammiert werden kann. Oft gibt es hier 16 Stufen, und meist bedeutet 0 den leisesten Ton.

Echte Musikerzeugung ist nur möglich, wenn der Computer mit einem Tongenerator ausgestattet ist. Es handelt sich hierbei um

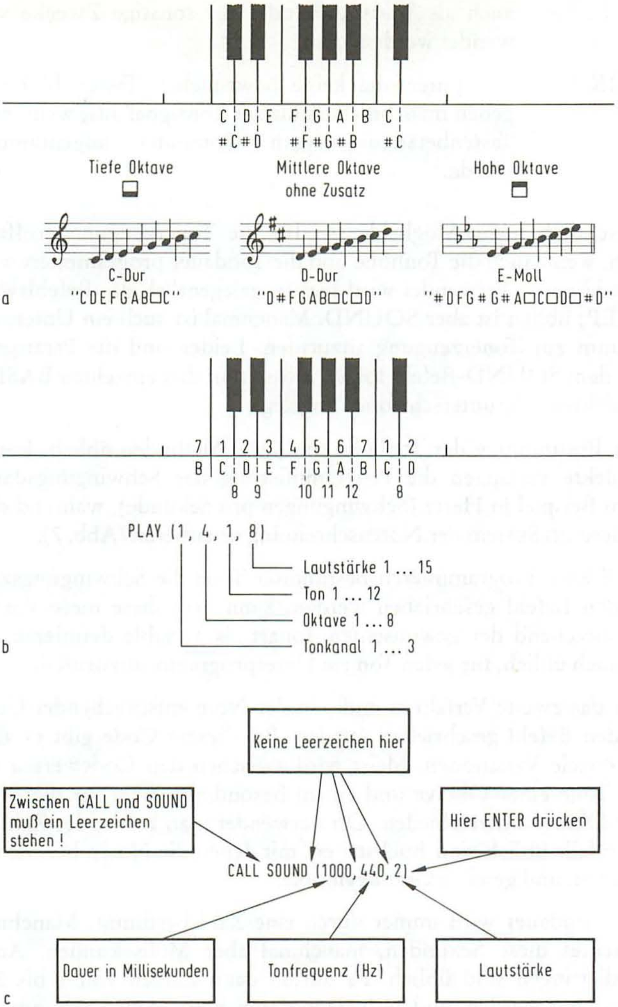


Abb. 7: Verschiedene Methoden für die Programmierung von Tönen
a. Einige Computer verwenden Befehle, die sich weitgehend an die Symbole der Notenschrift anlehnen.
b. Viele neuere BASIC-Dialekte arbeiten mit Codezahlen für die verschiedenen Parameter zur Erzeugung eines Tons.
c. Einige BASIC-Dialekte fordern die Eingabe der physikalischen Daten für die zu erzeugenden Töne

programmierbare hochintegrierte Bauelemente, die es erlauben, außer Tonhöhe, -länge und -dauer auch die physikalische Form der Tonschwingung zu programmieren. Hierdurch ist es möglich, die Klangfarbe eines Tons vorzugeben und damit zum Beispiel Orgel-, Flöten- und Gitarrentöne zu erzeugen.

Da Musik meist nicht in reinen Tönen, sondern in Akkorden erklingt, eröffnen hochwertige Tongeneratoren die Möglichkeit, mehrere Tonkanäle gleichzeitig erklingen zu lassen. Erst wenn diese Art der Programmierung vorgesehen ist, kann man seinen Computer zu echter Musikerzeugung verwenden. Hierzu ist dann neben der Beherrschung der Tonprogrammierung des benutzten Computers auch profundes Wissen in der Komponiertechnik erforderlich.

Wesentlich weniger komfortabel gestaltet sich die Musikerzeugung, wenn die Tongeneratoren nicht durch BASIC-Befehle anzusprechen sind. Dann müssen alle Werte (meist nach komplizierter Berechnung) mit POKE-Befehlen eingesetzt werden. Heute gibt es allerdings auch für die Computer, die derartig knapp ausgebaute BASIC-Versionen verwenden (z. B. den Commodore C-64), entsprechende Hilfsprogramme, mit denen einfache Musikprogrammierung möglich ist.

4. Das Laufverhalten verschiedener BASIC-Dialekte

Zu den Testprogrammen in diesem Kapitel

Mit den in diesem Kapitel gezeigten Testprogrammen kann geprüft werden, wie ein Computer bei bestimmten Befehlen arbeitet. Alle Testprogramme sind auf einem Genie-Computer mit voll ausgebautem Microsoft-BASIC geschrieben, das auch auf vielen anderen neuen Computern läuft. Wenn der zu testende Computer im Programmlauf einen Syntaxfehler meldet und das Programm abbricht, bedeutet dies, daß er eine Befehlsvariante nicht ausführen kann. Sie können dann das Programm mit

GOTO nächste Hunderter-Zeilennummer

weiterlaufen lassen. Es erfolgt danach eine Meldung, die sagt, was bei der verwendeten BASIC-Version nicht erlaubt ist. Wird der betreffende Befehl vom Computer angenommen, erfolgt kein Abbruch.

In den Testprogrammen stehende STOP-Befehle ermöglichen eine Beurteilung von Ergebnissen. Nach der BREAK-Meldung ist das Programm durch Eingabe des CONT-Kommandos fortzusetzen.

In einigen wenigen Fällen ist auch eine Abfrage im Direktbetrieb notwendig, auf die dann im Programm hingewiesen wird.

Da Anführungsstriche in Zeichenketten nicht erlaubt sind, wird an deren Stelle im Programm in einem auszugebenden Text CHR$(34) eingesetzt.

Die verschiedenen BASIC-Dialekte sind heute kaum noch zu zählen. Auch wenn die gleichen Befehlswörter verwendet werden, unterscheiden sie sich oft in Kleinigkeiten. Man kann aber erkennen, daß gewisse typische Merkmale bei mehreren Versionen übereinstimmen. In diesem Kapitel werden wesentliche Unterschiede erklärt, die es bei den BASIC-Typen bezüglich der Befehlsanordnung und des Laufverhaltens der Befehle gibt.

Es geht hier weniger um die Befehle selbst, sondern um ihre Wirkung im Programm. Da man diese nur schwer aus einem Handbuch ersehen kann, werden Testprogramme vorgestellt, mit denen jeder Computerbesitzer sein BASIC prüfen kann. Nur wer sein eigenes BASIC in allen Einzelheiten kennt, kann Programme in anderen Versionen verstehen und gegebenenfalls auf seinem Computer zum Laufen bringen.

4.1 Zeilenaufbau und Befehlsschreibung

Zeilennumerierung und Sprungziele

Im Gegensatz zu den meisten anderen Programmiersprachen arbeitet BASIC mit Zeilennummern, die vor dem Programmlauf in aufsteigender Reihenfolge sortiert werden. Dies hat den großen Vorteil, daß der Programmierer den Ablauf des Programms einfach ändern kann. Auf der anderen Seite werden BASIC-Programme leicht unübersichlich, wenn beim Programmieren nicht streng modular gearbeitet wird.

Da allgemein 2 Byte für die Speicherung der Zeilennummer als Binärzahl vorgesehen sind, lassen sich dort Zeilennummern von 0 bis 65535 unterbringen. In der Praxis sieht es allerdings meist etwas anders aus: Die Zeile 0 ist manchmal nicht zulässig, und die obersten Zeilennummern können oft ebenfalls nicht verwendet werden.

Kennzeichnend für die meisten Versionen ist es, daß nur im Programm verwendete Zeilennummern als Sprungziele benutzt werden dürfen, und daß als Zeilennummern keine Rechenausdrücke und damit auch keine Variablen erlaubt sind.

Von dieser Regel gibt es aber bemerkenswerte Ausnahmen. Es existieren auch BASIC-Versionen,

- die die Berechnung einer als Sprungziel verwendeten Zeilennummer gestatten, und/oder
- bei denen als Sprungziel auch eine Wortmarke (Label) oder eine Variable verwendet werden kann.

Die Möglichkeit, Sprungziele zu berechnen, die mit GOTO angesprungen werden sollen, findet man verhältnismäßig selten. Die Verwendung von Labels als Sprungziel wird in einem Normvorschlag für ein erweitertes BASIC gefordert, ist aber heute noch selten verwirklicht.

```
0 PRINT"Zeile 0 erlaubt
10 REM TEST1 Zeilennummerierung und GOTO
100 GOTO 230
200 PRINT"GOTO nur in besetzte Zeilen"
210 GOTO 400*2
300 PRINT"Keine Rechnungen in GOTO"
310 GOTO 900
400 PRINT"Zahl nach Rechenzeichen in GOTO
 wird nicht beachtet"
790 GOTO 900
800 PRINT"Rechnungen in GOTO erlaubt"
900 '
910 GOTO 1010
1000 PRINT"Kein ' statt REM erlaubt"
1010 PRINT"REM durch ' ersetzbar"
65512 PRINT"Letzte erlaubte Zeile 65512"

RUN
Zeile 0 erlaubt
Zeile fehlt  in 100
READY
>
>GOTO 200
GOTO nur in besetzte Zeilen
Zahl nach Rechenzeichen in GOTO
 wird nicht beachtet
REM durch ' ersetzbar
Letzte erlaubte Zeile 65512
READY
>
```

TEST 1, Zeilennumerierung und GOTO

Das Programm beginnt mit der Zeilennummer 0. Wenn dies nicht erlaubt ist, meldet der Computer einen Fehler, und das Programm muß mit GOTO 100 wieder gestartet werden. Dort wird geprüft, was geschieht, wenn der Computer einen Sprungbefehl zu einer nicht existierenden Zeilennummer findet. In 210 steht ein Befehl mit einem Rechenausdruck als Sprungziel, nach dem das Programm in verschiedener Weise weiterläuft oder abgebrochen wird. Anschließend wird getestet, ob REM durch das Hochkomma (Apostroph) ersetzt werden kann. Die letzte erlaubte Zeilennummer muß man ausprobieren.

Befehlsschreibung

Normalerweise werden in BASIC die Befehlswörter und die Variablen mit Großbuchstaben geschrieben. Es gibt aber auch Computer, wie zum Beispiel verschiedene Commodore-Modelle, bei denen Kleinbuchstaben für Befehle und Variable vorgeschrieben sind, wenn in der Betriebsart »Groß/Kleinschreibung« gearbeitet wird.

Es ist gute Programmierpraxis, vor und nach Befehlswörtern eine Leerstelle zu lassen. Einige Versionen setzen diese selbsttätig ein (Sinclair), andere verlangen sie. Letzteres ist meist der Fall, wenn auch längere Variablennamen zugelassen sind. Die meisten Versionen arbeiten aber sowohl bei »gepackter« Befehlsschreibung ohne Leerstellen als auch bei »ungepackter« Schreibweise mit Leerstellen.

Bei praktisch allen Versionen ist die Eingabe des Fragezeichens anstelle des PRINT-Befehls erlaubt. Viele Versionen gestatten auch Abkürzungen für andere Befehlswörter. Manchmal sind diese auf den Tasten vermerkt (Sinclair); manchmal dient ein Buchstabe mit nachfolgendem Punkt als Abkürzung (Commodore).

Zeilenlänge und Befehlsfolgen

Bei allen BASIC-Versionen ist die Länge der Befehlszeilen begrenzt. Manchmal sind nur 80 Zeichen zulässig; die neueren Versionen erlauben meist Zeilenlängen von ca. 240 bis 250 Zeichen. Gelegentlich gibt es auch eine Begrenzung auf eine bestimmte Anzahl von Bildschirmzeilen.

Bei der Zeilenlänge ist allerdings zu berücksichtigen, daß die BASIC-Befehlswörter meist nicht im Klartext, sondern als Bitmuster, den Dezimalzahlen von 0 bis 255 entsprechend, gespeichert werden. Man nennt diese Abkürzungen *TOKEN*. Sie sind bei allen BASIC-Dialekten unterschiedlich.

Auf alle Fälle gehören aber auch nicht auf dem Bildschirm sichtbare Steuerzeichen mit zu den für die Länge maßgeblichen Zeichen, wie zum Beispiel das Kommando für den Zeilentransport, wenn das Ende der Bildschirmzeile erreicht ist.

Im allgemeinen können mehrere Befehle, getrennt durch ein bestimmtes Satzzeichen, in eine Befehlszeile geschrieben werden. Als Trennzeichen wird fast immer der Doppelpunkt verwendet. Es gibt aber auch BASIC-Versionen, die den rückwärts geneigten Strich dafür verlangen.

Zeichenketten müssen in Programmbefehlen immer in Anführungszeichen stehen. Viele BASIC-Versionen stellen es allerdings frei, das letzte Anführungszeichen in der Zeile wegzulassen.

```
10 REM TEST2 Befehlsschreibung
100 PRINT CHR$(34);" am Zeilenende unnötig
110 GOTO 210
200 PRINT CHR$(34);" auch am Zeilenende nötig"
210 PRINT": als Befehlstrennung erlaubt":PRINT
220 GOTO 310
300 PRINT"kein : als Befehlstrennung"
310 PRINT"aaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccccccccccccccccccccc
dddddddddddddddddddddddddddddddddddddd
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
ffffffffffffffffffffffffffffffffffffff
ggggggggggggggggggggggggg

RUN
" am Zeilenende unnötig
: als Befehlstrennung erlaubt

aaaaaaaaaaaaaaaaaaaaaaaaaaaa
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
cccccccccccccccccccccccccccccccccccccc
dddddddddddddddddddddddddddddddddddddd
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
ffffffffffffffffffffffffffffffffffffff
ggggggggggggggggggggggggg
READY
```

TEST 2, Befehlsschreibung
Im Test wird geprüft, ob die Anführungsstriche am Ende einer Zeile entfallen können und ob der Doppelpunkt zur Befehlstrennung zulässig ist. Die erlaubte Länge einer Befehlszeile muß man ausprobieren. Sie kann unterschiedlich sein, je nachdem, ob die Zeile mitten im Programm oder am Ende steht. In der Befehlszeile müssen durch den Computer oft auch noch Hinweise auf die nächste zu verarbeitende Zeile gespeichert werden.

4.2 Variablennamen und Variablentypen

Variable, eigentlich symbolischer Namen für reservierte Speicherstellen, auf die im Programm zugegriffen werden kann, gibt es bei allen BASIC-Dialekten. Unterschiedlich sind allerdings die Regeln für die Namensgebung durch den Programmierer.

Grundsätzlich muß der Name immer mit einem Buchstaben beginnen. Meist können dann beliebige andere Zeichen (außer dem Zwi-

schenraum und den Satzzeichen) folgen. Allerdings werden vielfach nur die ersten 2 Zeichen als Namen vom Computer ausgewertet. Einige Dialekte berücksichtigen dagegen längere Namen, was die Übersichtlichkeit der Programme erhöht, aber auch Speicherplatz kostet.

Gelegentlich dürfen bestimmte Variable nur mit einem Buchstaben benannt werden (z. B. beim Sinclair).

```
10 REM TEST 3 Variablennamen
100 HANS = 7 : HAUS = 8
110 GOTO 210
200 PRINT"Variablenname zu lang"
210 PRINT "HANS =";HANS,"HAUS =";HAUS
220 IF HAUS = HANS THEN GOTO 250
230 PRINT"Alle Zeichen im Namen signifikant"
240 END
250 PRINT"HANS und HAUS nicht unterschieden"

RUN
HANS = 8          HAUS = 8
HANS und HAUS nicht unterschieden
READY
>
```

TEST 3, Variablennamen
Das Testprogramm meldet, wenn die mit 4 Buchstaben gekennzeichneten Variablen nicht angenommen werden. Wenn nur die ersten beiden Buchstaben für den Namen signifikant sind, unterscheiden sich die Variablennamen nicht, so daß der Inhalt von HANS gleich dem von HAUS ist.

Definition von Variablentypen

Alle voll ausgebauten BASIC-Dialekte kennen mindestens 3 Typen:

- Integervariable (Ganzzahlen), gekennzeichnet durch %
- Variable für Fließkommazahlen, ohne Kennzeichen oder mit !
- Zeichenkettenvariable, gekennzeichnet durch $.

Bei einigen Versionen gibt es dann noch zusätzlich
- Zahlenvariable mit doppelter Genauigkeit, Kennzeichen #.

Im allgemeinen muß das Variablenkennzeichen dem Namen unmittelbar folgen; ganz selten steht es vor dem Namen (z. B. beim ACORN ATOM).

Der Zahlenbereich, in dem Ganzzahlen verarbeitet werden können, liegt bei den meisten Computern zwischen −32768 und +32767.

Viele neuere Dialekte kennen besondere Definitionsbefehle, mit denen bestimmte Anfangsbuchstaben des Namens für einen Variablentyp reserviert werden können. Im weiteren Verlauf des Programms braucht dann das Typenkennzeichen bei diesen Variablen nicht angefügt zu werden. Bei der Definierung eines Typs einer Variablen gilt dieser meist für alle Zahlenvariablen, deren Name mit dem betreffenden Buchstaben anfängt, nicht jedoch für Zeichenkettenvariable, auch mit dem gleichen Buchstaben. Durch Anfügung eines Typenkennzeichens an einen Variablennamen kann meist die Definition wieder aufgehoben werden.

```
10 REM TEST 4 Definition von Variablentvpen
60 DEFINT A: PRINT"A als Integer definiert"
90 GOTO 120
100 PRINT"Keine Variablendefinierung möglich"
110 END
120 PRINT"LET A = 12324.56": A=12343.56
130 PRINT"1234.56 wird zu ";A
200 PRINT"Variable A kann nur Integer aufnehmen
300 A1 = 12.34
310 PRINT"Alle Zahlenvariablen mit dem gleichen
Anfangsbuchstaben sind Integervariable
320 PRINT"A1=12.34 wird zu ";A1
330 A$="HUMBOLDT"
340 PRINT"A$ = ";A$
350 PRINT"A und A$ sind verschieden"
360 END
400 PRINT"Definierung gilt für alle Variablen"

RUN
A als Integer definiert
LET A = 12324.56
1234.56 wird zu  12343
Variable A kann nur Integer aufnehmen
Alle Zahlenvariablen mit dem gleichen
Anfangsbuchstaben sind Integervariable
A1=12.34 wird zu  12
A$ = HUMBOLDT
A und A$ sind verschieden
READY
```

TEST 4, Definition von Variablentypen
Dieser Test ist nur sinnvoll, wenn der Computer die Definierung von Variablentypen erlaubt. Er untersucht, für welche Variablennamen eine Definierung jeweils gilt.

Erlaubte Länge von Zeichenketten

Die maximale Anzahl von Zeichen, die in einer Zeichenkette gespeichert werden können, ist unterschiedlich. Einige Dialekte erlauben nur ca. 80 Zeichen, während andere bis etwa 255 Zeichen zulassen.

Bei Zeichenketten wird oft der Inhalt in einem besonderen Zeichenspeicher getrennt von den Informationen über Namen und Länge gespeichert.

Für die Speicherung selbst sind drei Verfahren üblich:

1. Bei den BASIC-Versionen, die auf dem Mikroprozessor 6502 oder ähnlichen Typen laufen (z. B. Commodore, Apple), ist keine Vorbereitung für die Speicherung von Zeichenketten notwendig.
2. Bei vielen Versionen, die einen Z 80 als Zentraleinheit verwenden, muß der oberste Teil des RAM mit dem CLEAR-Befehl und nachfolgender Zahl als Zeichenspeicher reserviert werden (z. B. TRS 80 und Genie).
3. Es gibt auch BASIC-Dialekte, die Zeichenketten als eindimensionale Felder speichern. Bei ihnen muß die Länge jedes im Programm verwendeten Strings vorher mit dem DIM-Befehl festgelegt werden (z. B. Sinclair und Hewlett-Packard).

Der Vorteil der Zeichenkettenspeicherung bei den ersten beiden Verfahren (der darin liegt, daß die Länge der Ketten während des Programms im Rahmen der erlaubten Maximallänge auch beliebig

```
10 REM TEST 5 Stringlänge
100 A$ = "X"
110 FOR K = 1 TO 100
120 B$ = B$+A$
130 NEXT: GOTO 220
200 PRINT"Reservierung mit CLEAR notwendig"
210 CLEAR 1000: A$ = "X"
220 FOR K = 1 TO 300
230 B$ = B$+A$
240 NEXT
300 PRINT"Erlaubte Stringlänge =";LEN(B$)

RUN
Zuwenig Textspeicher in 120
READY
>
>GOTO 200
Reservierung mit CLEAR notwendig
Text$ zu lang in 230
READY
>
>GOTO 300
Erlaubte Stringlänge = 255
READY
```

TEST 5, Stringlänge
Das Programm untersucht, ob eine Reservierung für die Zeichenspeicherung notwendig ist und wie lang dann Zeichenketten äußerstenfalls sein dürfen.

erweitert werden kann) muß durch die gelegentlich notwendige Aufräumung des Speichers erkauft werden.

Mit dem englischen Wort GARBAGE COLLECTION wird eine Neuordnung des Zeichenspeichers bezeichnet, die der Computer gelegentlich mitten im Programmlauf durchführt, wenn er keinen Platz mehr im Zeichenspeicher findet. Erst bei diesem Vorgang, der einige Sekunden dauern kann, werden die nicht mehr aktuellen Daten dort gelöscht; denn bei jeder neuen Speicherung einer auch schon vorher verwendeten String-Variablen wird neuer Platz im Zeichenspeicher belegt.

Ein Speicheraufräumen ist bei dem dritten Verfahren nicht notwendig. Dafür ergibt sich der Nachteil, daß im Programmlauf nur Zeichenketten bis zur dimensionierten Länge gespeichert werden können.

Speicherbedarf für Variable

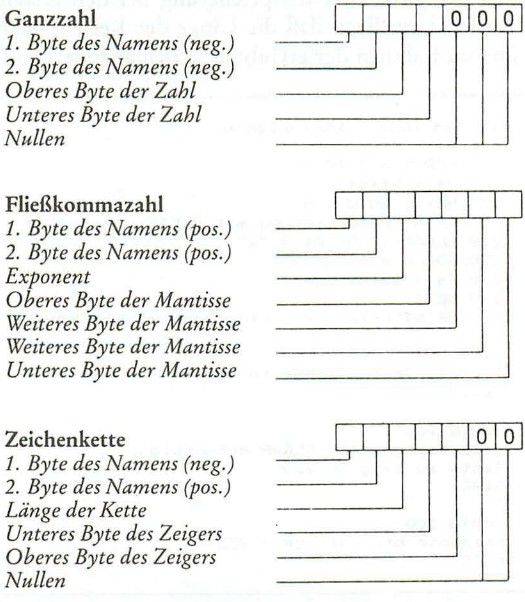


Abb. 8: Aufbau der Variablenspeicher bei Apple und Commodore
Grundsätzlich werden für alle Variablentypen immer 7 Byte beansprucht und die nicht benötigten Byte mit Nullen aufgefüllt. Den Typ der gespeicherten Variablen erkennt der Computer an der Art, wie die zwei Zeichen des Namens gespeichert sind

Variable werden bei den verschiedenen BASIC-Versionen unterschiedlich gespeichert. Normalerweise braucht sich der Programmierer hierum nicht zu kümmern. Wenn man den Speicherbedarf für einen Programmlauf schätzen will, muß man aber doch wissen, wie viele Byte jeweils für die Speicherung einer Variablen benötigt werden.

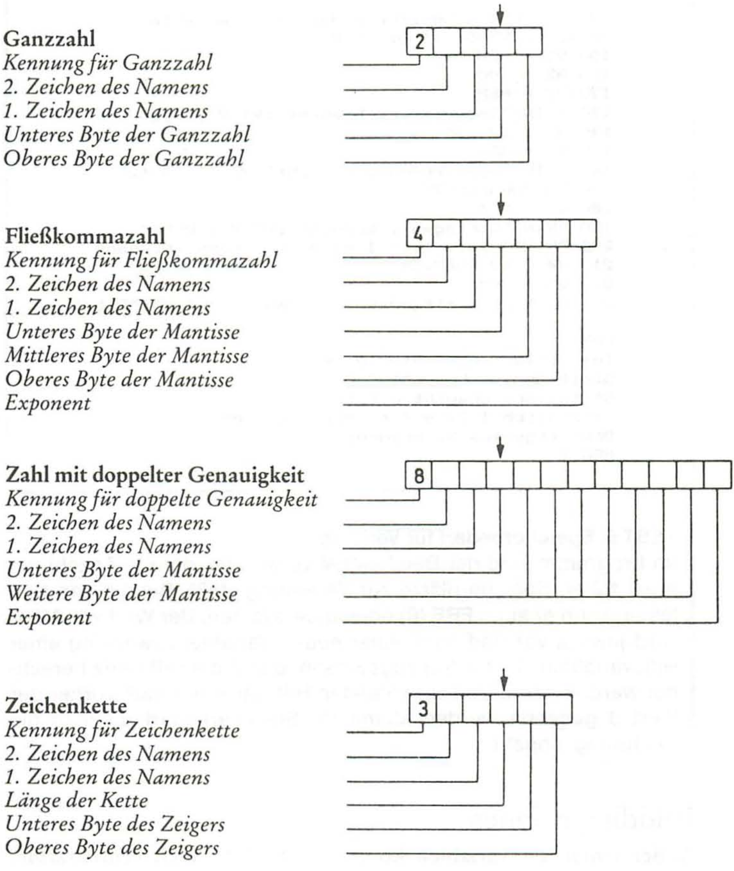


Abb. 9: Aufbau der Variablenspeicher beim TRS 80 und Genie
Für die verschiedenen Variablentypen wird unterschiedlicher Speicherraum verwendet. Der Typ ist an der als erstes gespeicherten Typenziffer zu erkennen

Neben dem eigentlichen Inhalt der Variablen muß auch immer der Variablenname und ein Hinweis auf den Variablentyp mitgespeichert werden. Hierfür gibt es recht unterschiedliche Verfahren bei den BASIC-Dialekten. Sie sollen hier nicht alle im einzelnen erklärt werden, da sie für das Verständnis fremder Programme nicht wesentlich sind. Zwei verbreitete Verfahren sind in den Abb. 8 und 9 dargestellt.

```
10 REM TEST 6 Speicherbedarf für Variable
90 V2=0: V3=0: V4=0: V5=0
100 V1 = MEM
110 A% = 1000
120 V2 = MEM
130 PRINT"Integervar. braucht";V1-V2;"Byte
140 B! = 123.45
150 V3 = MEM
160 PRINT"Gleitkommav. braucht";V2-V3;"Byte"
170 C$="HUMBOLDT"
180 V4 = MEM
190 PRINT"Stringvar, braucht";V3-V4;"Byte"
200 PRINT" zuzüglich 1 Byte für jedes Zeichen"
210 D# = 1234567.89
220 V5 = MEM
230 PRINT"Doppeltgenaue V. braucht";V4-V5;"Byte"

RUN
Integervar. braucht 5 Byte
Gleitkommav. braucht 7 Byte
Stringvar, braucht 6 Byte
 zuzüglich 1 Byte für jedes Zeichen
Doppeltgenaue V. braucht 11 Byte
READY
```

TEST 6, Speicherbedarf für Variable
Im Programm wird der Befehl **MEM** verwendet, der die Anzahl der noch freien Speicherplätze zur Verfügung stellt. Bei einigen Dialekten kann er auch **FRE (0)** oder anders lauten. Der Wert von MEM wird jeweils vor und nach einer neuen Variablenzuweisung einer Hilfsvariablen (V1 bis V5) zugewiesen, damit die Differenz berechnet werden kann. Den verwendeten Hilfsvariablen muß vorher der Wert 0 gegeben werden, damit ihr Speicherbedarf nicht in die Rechnung eingeht.

Felddimensionen

Außer einfachen Variablen können in BASIC auch *Feldvariable*, manchmal auch *Bereichsvariable* genannt, verwendet werden. Bei ihnen kann der Variablenname jeweils noch durch eine oder mehrere Indexziffern ergänzt werden. Es gibt Dialekte (z. B. Sinclair), bei denen die Feldnamen nur aus einem Buchstaben bestehen dürfen. Meist sind aber auch zwei Zeichen zulässig.

Felder müssen immer vor der Verwendung mit dem DIM-Befehl dimensioniert werden. Sehr viele BASIC-Dialekte gestatten aber die Benutzung eindimensionaler Felder ohne vorherige Dimensionierung, wenn nur die Indexziffern 0 bis 10 Verwendung finden. Bei einigen Versionen kann mit **OPTION BASE** bestimmt werden, ob das erste Feldelement den Index 0 oder 1 erhält.

Die Anzahl der Feldelemente, also der Indexziffern, ist meist nicht beschränkt, sondern nur von der Größe des Arbeitsspeichers abhängig.

Unzulässig ist bei den meisten BASIC-Dialekten das Ansprechen der DIM-Befehle oder deren Änderung im weiteren Programmlauf. Bei einigen Dialekten gibt es aber hierfür den Befehl **ERASE,** mit dem alle Dimensionierungen oder die Dimensionierungen in einer bestimmten Zeile aufgehoben werden können.

```
10 REM TEST 7 Felddimensionierung
100 X=MEM: PRINT X;"Vor Dimensionierung"
110 DIM A(80,100)
120 Y=MEM: PRINT Y;"Nach Dimensionierung"
140 PRINT"Zahlenfeld mit 8000 Elementen"
150 PRINT"benötigt";X-Y;"Speicherplätze"
160 DIM B(10)
170 PRINT MEM
180 DIM B(20)
190 END
200 PRINT"Änderung einer Dimensionierung"
210 PRINT"        ist verboten"

RUN
 36654 Vor Dimensionierung
 3913 Nach Dimensionierung
Zahlenfeld mit 8000 Elementen
benötigt 32741 Speicherplätze
 3865
Neue Felddimension in 180
READY
>
>GOTO 200
Änderung einer Dimensionierung
        ist verboten
READY
>
```

TEST 7, Felddimensionierung
Im Programm wird ein zweidimensionales Feld mit 80 Spalten und 100 Zeilen dimensioniert, was ca. 32 KByte Speicherraum erfordert. Eine Redimensionierung eines schon dimensionierten Feldes ist in der verwendeten BASIC-Version nicht zulässig.

Wertzuweisung an Variable

Bei fast allen BASIC-Versionen kann die Wertzuweisung auch allein durch das Gleichheitszeichen erfolgen. Nur wenige Dialekte (z. B. Sinclair) fordern das Befehlswort LET.

Unterschiede gibt es allerdings bei der Behandlung leerer Variablen im Programm. Bei einigen Dialekten muß einer Variablen, ehe sie in einem Rechenausdruck oder PRINT-Befehl verwendet wird, ein Wert (der auch 0 sein kann) zugewiesen werden. Neuere Versionen verlangen dies nicht mehr. Einer leeren Zahlenvariablen wird automatisch der Wert 0 zugewiesen, während eine leere Zeichenkettenvariable oft ignoriert wird.

```
10 REM TEST 8 Leere Variable
100 PRINT A
110 PRINT"Bei leerer Zahlenvariablen wird"
120 PRINT ASC(STR$(A));"als Code eingesetzt"
130 GOTO 230
200 PRINT"Variable darf nicht ohne vorherige"
210 PRINT"Zuweisung verwendet werden"
220 END
230 PRINT A$
240 PRINT"Bei leerer Stringvariablen wird"
250 PRINT ASC(A$);"als Code eingesetzt"
260 END
300 PRINT"Kein abfragbarer Code eingesetzt"

RUN
 0
Bei leerer Zahlenvariablen wird
 32 als Code eingesetzt

Bei leerer Stringvariablen wird
Unerlaubter Funktionsaufruf in 250
READY
>
>GOTO 300
Kein abfragbarer Code eingesetzt
READY
>
```

TEST 8, leere Variable

Das Programm untersucht, was geschieht, wenn eine Variable im Programm verwendet wird, ohne daß ihr vorher ein Wert zugewiesen wurde.

Bei den meisten BASIC-Dialekten muß jede Wertzuweisung an eine Variable einzeln programmiert werden. Nur wenige Versionen erlauben Mehrfachzuweisungen in einem Befehl.

```
10 REM TEST 9 Mehrfachzuweisungen
100 A=B=10
110 IF B=10 THEN 300
120 PRINT"A =";A:"B =";B
200 PRINT"Mehrfachzuweisungen unzulässig
210 END
300 PRINT"Mehrfachzuweisungen zulässig

RUN
A = 0 B = 0
Mehrfachzuweisungen unzulässig
READY
```

TEST 9, Mehrfachzuweisungen
Dieser Test zeigt, daß bei dem verwendeten BASIC Mehrfachzuweisungen zwar nicht zulässig sind, aber doch nicht zum Programmabbruch führen.

4.3 Zahlenverarbeitung

Im Programm zu verwendende Zahlen werden in BASIC immer als Dezimalzahlen eingegeben und auch in gleicher Weise vom Programm ausgegeben. Da die Zahlen intern aber binär verarbeitet und gespeichert werden, ist nach jeder Eingabe und vor jeder Ausgabe eine Umrechnung notwendig, die BASIC selbsttätig durchführt. Einige sich dabei ergebende Besonderheiten können bei den verschiedenen BASIC-Dialekten zu unterschiedlichen Wirkungen führen.

Zunächst gibt es schon verschiedenes Verhalten bei der Zahleneingabe und Interpretierung. Benutzerfreundliche Versionen ignorieren alle Leerstellen innerhalb einer Ziffernfolge und ziehen diese zu einer Zahl zusammen.

```
10 REM TEST 10 Leerstellen in Zahlen
100 A = 2  3 4: PRINT"A = 2  3 4"
110 IF A=234 THEN PRINT A: GOTO 140
120 PRINT"Leerstellen innerhalb von"
130 PRINT"Zahlen nicht erlaubt": END
140 PRINT"Leerstellen innerhalb von
150 PRINT"Zahlen werden ignoriert"

RUN
A = 2  3 4
 234
Leerstellen innerhalb von
Zahlen werden ignoriert
READY
```

TEST 10, Leerstellen in Zahlen
Der Test zeigt, ob mehrere durch Leerstellen getrennte Ziffern zu einer Zahl zusammengezogen werden.

Zahlen werden in allen BASIC-Dialekten nur mit einer beschränkten Ziffernzahl (Stellenzahl) verarbeitet und ausgegeben. Hierbei sind zwei Arbeitsweisen üblich:

- Viele BASIC-Versionen arbeiten intern mit bis zu 32stelligen Binärzahlen. Ausgegeben werden dann meist Dezimalzahlen mit 9 Stellen mit Rundung aus der 10. Stelle.
- Neuere BASIC-Versionen verarbeiten Fließkommazahlen in zwei Genauigkeiten. Bei einfacher Genauigkeit werden 6 Stellen mit Rundung aus der 7. Stelle ausgegeben, während bei doppelter Genauigkeit 15 Dezimalstellen genau gerechnet werden. Wenn mit doppelter Genauigkeit gearbeitet werden soll, müssen die verwendeten Variablen mit dem Doppelkreuz (#) gekennzeichnet oder vorher als doppelt genau definiert werden.

Müssen Zahlen verarbeitet werden, die kleiner oder größer sind, als auf diese Weise darstellbar, wird zur *Exponentialdarstellung* (wissenschaftliche Notation) übergegangen.

```
10 REM TEST 11 Zahlendarstellung
90 A=123456: A#=123456
100 FOR K = 0 TO 13
110 PRINT A* 10ÄK. A#* 10ÄK
120 NEXT

 123456              123456
 1.23456E+06         1234560.117736816
 1.23456E+07         12345600.94189453
 1.23456E+08         123456007.5351563
 1.23456E+09         1234560120.5625
 1.23456E+10         12345601929
 1.23456E+11         123456023148
 1.23456E+12         1234559629632
 1.23456E+13         12345608888832
 1.23456E+14         123455968395264
 1.23456E+15         1234560379256832
 1.23456E+16         1.23455916563497D+16
 1.23456E+17         1.23456039948386 3D+17
 1.23456E+18         1.234561014385607D+18
READY
```

TEST 11, Zahlendarstellung
Der Test zeigt, wann der Übergang zur Exponentialdarstellung geschieht. Man erkennt auch, wie hier die Rundung aus der 7. Stelle erfolgt. Wenn die verwendete BASIC-Version kein Rechnen mit doppelter Genauigkeit kennt, müssen die betreffenden Befehle weggelassen werden.

Schon bei diesem Test zeigt sich, daß die bei dem verwendeten BASIC benutzten Algorithmen zum Rechnen und zur Zahlenwandlung bei großen Zahlen nicht korrekt arbeiten. Dies ist ein Übel, das sehr viele BASIC-Dialekte aufweisen. Es kommt hinzu, daß das Laufverhalten nur sehr schwer vorauszusagen ist, wenn mehrere Rechnungen und Wandlungen hintereinander folgen. Einige neue BASIC-Versionen rechnen darum intern nicht mehr mit reinen Binärzahlen, sondern im *BCD-System* (Binary Coded Decimal), was immer genaue Rechnung ergibt.

```
10 REM TEST 12 Genauigkeit
90 A=100000: A#=100000
100 FOR K = 0 TO 12
110 PRINT A* 10^K, A#* 10^K
120 NEXT


RUN
 100000            100000
 1E+06             1000000.095367432
 1E+07             10000000.76293945
 1E+08             100000006.1035156
 1E+09             1000000097.65625
 1E+10             10000001562.5
 1E+11             100000018750
 1E+12             999999700000
 1E+13             10000007200000
 1E+14             99999974400000
 1E+15             1000000307200000
 9.99999E+15       9999993241600000
 1E+17             1.000000323584D+17
READY
```

TEST 12, Genauigkeit
Dieser Test, dessen Ergebnisse leicht nachzurechnen sind, deckt erstaunliche Fehler auch bei doppeltgenauer Rechnung auf. Verschiedene BASIC-Versionen laufen hier anders.

Die meisten Rechenalgorithmen enthalten auch Anweisung für die Rundung, da meist mit mehr Stellen gerechnet wird, als angezeigt werden. Dadurch bleiben Rechenergebnisse, soweit sie nicht in die Nähe von Null kommen, doch ausreichend zuverlässig.

```
10 REM TEST 13 Rundung
90 A=2: E=3
100 FOR K = 1 TO 200 STEP 20
110 A=A*K
120 B=A/3: C=B*3
130 PRINT A,B.C
140 NEXT
150 STOP
200 FOR K = 1 TO 10
210 E=E*K
220 F=E/3+K: G=(F-K)*3
230 PRINT E,F,G
240 NEXT

RUN
 2                .666667          2
 42               14               42
 1722             574              1722
 105042           35014            105042
 8.5084E+06       2.83613E+06      8.5084E+06
 8.59349E+08      2.8645E+08       8.59349E+08
 1.03981E+11      3.46604E+10      1.03981E+11
 1.46613E+13      4.88712E+12      1.46613E+13
 2.36048E+15      7.86826E+14      2.36048E+15
 4.27246E+17      1.42415E+17      4.27246E+17
Break in 150
READY
>

CONT
 3                2                3
 6                4                6
 18               9                18
 72               28               72
 360              125              360
 2160             726              2160
 15120            5047             15120
 120960           40328            120960
 1.08864E+06      362889           1.08864E+06
 1.08864E+07      3.62881E+06      1.08864E+07
READY
>
```

TEST 13, Rundung
Ob die Rundung bei entgegengesetzten Rechenverfahren immer in gleicher Weise arbeitet, kann mit diesem Test geprüft werden. Es werden Zahlen zunächst geteilt; dann wird das Ergebnis mit dem Divisor wieder multipliziert. Wenn die Rundung richtig arbeitet, müssen die ursprüngliche Zahl und das Ergebnis der Multiplikation übereinstimmen.
Im zweiten Teil wird geprüft, ob eine kleine Veränderung des Quotienten durch Addition einer Zahl ebenso durch Subtraktion der gleichen Zahl vor der Division richtig ausgeführt wird.

Rundungs- oder, besser gesagt, Rechenfehler gibt es aber meist bei sehr kleinen Zahlen. Diese Fehler können dazu führen, daß Vergleiche als falsch erkannt werden, auch wenn rechnerisch ein wahres Vergleichsergebnis vorliegt.

```
10 REM TEST 14 Rundungsfehler
90 A=1: B=.5: C=.8: D=.2
100 A=A-B: C=C-D
110 PRINT A,C,CDBL(C)
120 IF C= -.8 THEN 180
140 IF C< -.8 THEN 200
150 GOTO 100
180 PRINT"Zahlengleichheit bei -.8 erkannt: END
200 PRINT"Zahlengleichheit bei -.8 nicht erkannt

RUN
 .5              .6              .6000000238418579
 0               .4              .4000000357627869
-.5              .2              .2000000327825546
-1               2.98023E-08     2.980232238769531D-08
-1.5            -.2             -.1999999731779099
-2              -.4             -.3999999761581421
-2.5            -.6             -.5999999642372131
-3              -.8             -.7999999523162842
-3.5            -1              -.9999999403953552
Zahlengleichheit bei -.8 nicht erkannt
READY
```

TEST 14, Rundungsfehler
Dieser Test zeigt, wann der Nulldurchgang noch erkannt wird, wenn von der Ausgangszahl in jedem Schleifenlauf kleine Zahlen abgezogen werden. Mit der Funktion **CDBL** wird das Rechenergebnis in doppelte Genauigkeit gewandelt, wodurch man erkennt, welches interne Rechenergebnis entstanden ist.
Die nur mit einem Gleichheitszeichen formulierte Bedingung wird nicht als wahr erkannt, da der beim Nulldurchgang entstandene Fehler immer weiter mitgeschleppt wird.

Die Wandlung von einer Genauigkeitsdarstellung in die andere funktioniert nicht immer so, wie im Testbeispiel gezeigt.

```
10 REM TEST 15 Doppelt genau
100 A!=123/456
110 PRINT"123/456 einfach gerechnet";A!
120 B# =123/456
130 PRINT"doppelt genau gerechnet";B#
140 PRINT"in dopp.genau gewandelt";CDBL(A!)
150 PRINT"in einf.genau zurück";CSNG(B#)

RUN
123/456 einfach gerechnet .269737
doppelt genau gerechnet .2697368562221527
in dopp.genau gewandelt .2697368562221527
in einf.genau zurück .269737
READY
```

TEST 15, Doppelt genau
In diesem Test wird untersucht, ob es möglich ist, das Ergebnis einer einfach genauen Rechnung in doppelte Genauigkeit zu wandeln, bzw. umgekehrt.

Die in diesem Kapitel erklärten Tests zeigen, daß man bei einigen BASIC-Versionen den mit vielen Stellen ausgegebenen Rechenergebnissen nicht unbedingt trauen kann. Es ist darum zu empfehlen, Rechnungen gegebenenfalls mit einfachen, leicht nachprüfbaren Zahlen auszuprobieren.

4.4 Wirkung der Funktionen

Die Wirkungsweise der Funktionsbefehle ist mit wenigen Ausnahmen in allen BASIC-Versionen gleich. Da die einzelnen Dialekte aber zum Teil verschiedene Algorithmen zur Berechnung der Funktionswerte verwenden, können schon Unterschiede bei den Ergebnissen auftreten.

Winkel- und Rechenfunktionen

Nur wenige BASIC-Dialekte bieten dem Benutzer die Wahl, ob er den Winkel in Grad oder als Radiant einsetzen will. Im allgemeinen muß der Winkel im Bogenmaß in das Argument der Funktion geschrieben werden.

Nur in wenigen BASIC-Dialekten sind alle bekannten Winkelfunktionen verfügbar. Immer besteht jedoch die Möglichkeit, nicht

```
10 REM TEST 16 Sinus/Cosinus
90 PI=3.14159
100 FOR K = 0 TO 90 STEP 10
110 G = K*2*PI/360
120 PRINT SIN(G),COS(G)
130 NEXT

RUN
 0               1
 .173648         .984808
 .34202          .939693
 .5              .866026
 .642787         .766045
 .766044         .642788
 .866025         .500001
 .939692         .342021
 .984808         .173649
 1               1.12352E-06
READY
```

TEST 16, Sinus/Cosinus
Der Test soll zeigen, bei welchen Winkeln der verwendete Rechenalgorithmus ungenaue Ergebnisse liefert. Es ist zweckmäßig, diesen Test auch mit der Schrittweite 1 in Zeile 100 laufen zu lassen.

vorhandene Winkelfunktionen durch Näherungsformeln zu berechnen.

Ungenauigkeiten treten bei vielen BASIC-Dialekten auf, wenn der Funktionswert sehr klein wird.

Ähnliche Ungenauigkeiten passieren bei anderen Funktionen, deren Ergebnis auf einer Reihenbildung beruht.

Zufallszahlen

Große Unterschiede gibt es bei der Funktion RND zur Erzeugung von Zufallszahlen. In den meisten Dialekten werden nur Zahlen zwischen 0 und 1 erzeugt; die Art der Zahlenfolge kann aber unterschiedlich sein, je nachdem, ob als Argumente 0, eine positive oder eine negative Zahl eingesetzt wird.

Nur wenige BASIC-Dialekte erlauben die Erzeugung größerer ganzzahliger Zufallszahlen ohne zusätzlichen Rechenvorgang. Beim TRS 80 und beim Genie werden, wenn eine Ganzzahl als Argument eingesetzt wird, Zufallszahlen zwischen 1 und dieser Zahl gebildet.

Eine Reihe von Computern erzeugt immer die gleiche Folge von Zufallszahlen. Es gibt dann aber besondere Befehle, mit denen der Zufallsgenerator vor jedem Programmlauf auf einen anderen

```
10 REM TEST 17 Zufallszahlen
20 ON ERROR GOTO 200
100 FOR K = 1 TO 10
110 PRINT RND(0),RND(5),RND(-1)
120 NEXT
200 RESUME 120

RUN
 .903326        4              .349111        2
 .286094        3              .872227        4
 .671405        2              .906889        2
 .476437        3              .197911        1
 .504106        5              .555534        3
```

TEST 17, Zufallszahlen
Das Programm prüft, was geschieht, wenn verschiedene Argumente in die RND-Funktion gesetzt werden. Da das Testprogramm auf einem Genie-Computer lief und dieser negative Zahlen als Argument nicht annimmt, enthält es eine Fehlerbehandlungsroutine. Sie verhindert den Programmabbruch, wenn ein Syntaxfehler (RND mit negativem Argument) gefunden wird. Zeile 20 und Zeile 200 können bei anderen Computern entfallen.

Anfangswert gesetzt werden kann. Die Befehlswörter **RANDOM** oder **RANDOMIZE** sind hierfür üblich. Bei Computern, die eine immer mitlaufende Uhr besitzen (z. B. Commodore), kann auch die Uhrzeit als Anfangswert verwendet werden, wodurch sich unterschiedliche Folgen von Zahlen ergeben.

Wenn dieser Test mehrfach durchgeführt wird, ist auch gut zu erkennen, ob immer die gleiche Folge von Zufallszahlen läuft.

Zeichenkettenfunktionen

Auch die Zeichenkettenfunktionen arbeiten in allen BASIC-Dialekten im wesentlichen gleich, nur sind nicht immer alle verfügbar.

Einen wesentlichen Unterschied gibt es lediglich bei der STR$-Funktion zur Wandlung einer Zahl in eine Zeichenkette. In verschiedenen Dialekten wird bei der Wandlung einer positiven Zahl in eine Kette immer eine Leerstelle als erstes Zeichen in die Zeichenkettenvariable gesetzt. Dies muß gegebenenfalls bei Vergleichen mitberücksichtigt werden.

```
10 REM TEST 18 Funktionen  STR$ und VAL
90 PRINT"A=+5 und B$=STR$(A)"
100 A=+5: B$=STR$(A)
110 PRINT"Länge von B$=";LEN(B$);"Byte
120 PRINT"1.Byte hat ASCII-Wert";ASC(B$)
130 PRINT"2.Byte hat ASCII-Wert";ASC(RIGHT$(B$,1))
140 PRINT"Bei VAL(B$)=5 wird gemeldet
150 IF VAL(B$)=5 THEN PRINT "WAHR": GOTO 200
160 PRINT"FALSCH"
200 PRINT"Bei STR$(A)=";CHR$(34);"5";CHR$(34);
"wird gemeldet
210 IF STR$(A)="5" THEN PRINT"WAHR": END
220 PRINT"FALSCH"

RUN
A=+5 und B$=STR$(A)
Länge von B$= 2 Byte
1.Byte hat ASCII-Wert 32
2.Byte hat ASCII-Wert 53
Bei VAL(B$)=5 wird gemeldet
WAHR
Bei STR$(A)="5"wird gemeldet
FALSCH
READY
```

TEST 18, Funktionen STR$ und VAL
Der Test zeigt, ob bei der Wandlung einer Zahl in eine Zeichenkette nur die Zahl oder vorweg noch ein anderes Zeichen in die String-Variable gesetzt wird. Der Test offenbart auch, wie Vergleiche behandelt werden.

Sonstige Funktionen

In dieser Gruppe ist die Syntax für die Anwendung der TAB-Funktion gelegentlich unterschiedlich. Auch BASIC-Dialekte, die Leerstellen grundsätzlich tolerieren, verlangen oft, daß zwischen dem Befehlswort TAB und der öffnenden Klammer für das Argument kein Zwischenraum stehen darf. Es besteht auch keine Einigkeit darüber, ob nach der TAB-Funktion ein Semikolon stehen muß.

```
10 REM TEST 19 TAB-Funktion
100 PRINT TAB (10) "Hallo":
110 IF POS(1) < 10 THEN 200
200 PRINT
210 PRINT"Hinter TAB keine Leerstelle erlaubt
220 PRINT TAB(20) "Guten Morgen"
230 PRINT"Semikolon nach TAB kann entfallen
240 END
300 PRINT"Nach TAB Semikolon erforderlich
310 PRINT TAB(10);"Hallo"

RUN
 0 Hallo
Hinter TAB keine Leerstelle erlaubt
                    Guten Morgen
Semikolon nach TAB kann entfallen
READY
```

TEST 19, TAB-Funktion

In diesem Test verwenden wir die nicht immer zur Verfügung stehende **POS**-Funktion, die mit einem beliebigen Argument die augenblickliche Cursorspalte ausgibt. Es soll überprüft werden, ob TAB auch mit nachfolgender Leerstelle angenommen wurde.

4.5 Verschiedene Eingabebefehle

Eingabebefehle ermöglichen es dem Bediener, während des Programmlaufs Daten einzugeben. In allen BASIC-Versionen gibt es mindestens zwei Eingabebefehle. Beide haben Vor- und Nachteile.

INPUT

Der INPUT-Befehl ist praktisch in allen voll ausgebauten Dialekten in gleicher Weise anwendbar. Grundsätzlich werden durch diesen Befehl Tastatureingaben der im Befehl stehenden Variablen zugewiesen. Vorteilhaft ist es, daß man bei den meisten BASIC-Versionen die Erklärung für den Benutzer, wofür die Eingabe angefordert wird, als Zeichenkette in den Befehl schreiben kann. Man spart so einen besonderen PRINT-Befehl.

Jede Tastatureingabe wird sofort auf dem Bildschirm gezeigt. Falsche Zeichen können mit der Löschtaste *(BACKSPACE)* wieder gelöscht werden.

Der Computer überprüft meist, ob die Eingabe zur im Befehl stehenden Variablen paßt. Er fordert mit dem Wort *REDO* zu erneuter Eingabe auf, wenn man versucht, einen Buchstaben in eine Zahlenvariable zu schreiben.

Da das Komma als Trennzeichen zwischen zwei Eingaben verwendet wird, kann man es mit diesem Befehl nicht eingeben. Das gleiche gilt meist für den Doppelpunkt und die Anführungszeichen.

Einige ältere BASIC-Versionen (z. B. cbm) haben die unangenehme Eigenschaft, das Programm abzubrechen, wenn nach einem INPUT-Befehl nur die RETURN-Taste allein betätigt wird. Leere INPUT-Eingaben müssen hier vermieden oder durch besondere Befehle abgefangen werden.

```
10 REM TEST 20 INPUT-Befehl
20 INPUT"Komma eingeben":A
30 PRINT" Komma wurde eingegeben",A
40 INPUT"Doppelpunkt eingeben":A
50 PRINT"Doppelpunkt eingegeben";A
60 INPUT"Nur CR betätigen";A
70 PRINT"Leerer INPUT erlaubt"
80 GOTO 110
100 PRINT"Kein leerer INPUT erlaubt"
110 INPUT"Anführungstriche im Text":A$
120 PRINT A$
130 END
200 PRINT"Keine Erklärung im INPUT-Befehl"

RUN
Komma eingeben? ,
?Extra ignored
 Komma wurde eingegeben              0
Doppelpunkt eingeben? :
?Extra ignored
Doppelpunkt eingegeben 0
Nur CR betätigen?
Leerer INPUT erlaubt
Anführungstriche im Text? "ee"ff"gg"
?REDO
Anführungstriche im Text? ee"ff"gg
ee"ff"gg
READY
```

TEST 20, INPUT-Befehl

Der Test soll zeigen, ob es beim INPUT-Befehl Abweichungen vom Üblichen gibt. Probieren Sie, ob ein Programmabbruch erfolgt, wenn nach dem Erscheinen des INPUT-Fragezeichens nur die RETURN-Taste betätigt wird. Auch sollte untersucht werden, was bei der Eingabe des Kommas, des Doppelpunkts und der Anführungszeichen geschieht.

GET oder INKEY$

Der zweite Befehl, der für Eingaben verwendet werden kann, ist ein Abfragebefehl für die Tastatur, der bei den verschiedenen BASIC-Versionen unterschiedlich einzusetzen ist, und für den auch zwei verschiedene Befehlswörter gebräuchlich sind. Grundsätzlich wird mit diesem Befehl der Tastencode einer betätigten Taste sofort einer Zeichenkettenvariablen zugewiesen, ohne daß die RETURN-Taste zu betätigen ist.

```
10 REM TEST 21 Einqabebefehle
20 GET A$
30 PRINT A$;
40 IF A$=CHR$(13) THEN 20
50  END
100 GET A$: IF A$="" THEN 100
110 PRINT A$;
120 IF A$=CHR$(13) THEN 100
130  END
200 A$=INKEY$: IF A$="" THEN 200
210 PRINT A$;
220 IF A$=CHR$(13) THEN 240
230 GOTO 200
240  END

RUN
Tvpabweichunq in 20
READY
>
>GOTO 100
Typabweichunq in 100
READY
>
>GOTO 200
1234qwert"#$%&',./;:
READY
>
```

TEST 21, Eingabebefehle
Mit dem Test kann untersucht werden, welche Eingaben ein Computer akzeptiert. Dieser Befehl läuft dann in einer Schleife, bis man die RETURN-Taste betätigt. Man kann also ausprobieren, welche Zeichen angenommen werden.

Da die Tastenabfrage sehr schnell geht, kann der Computer sie nur erkennen, wenn das Tastenfeld in einer Schleife vielfach abgefragt wird. Meist muß diese Warteschleife programmiert werden. Einige Computer (z. B. Apple) gehen aber auch selbst nach dem GET-Befehl in Wartestellung, bis eine Taste betätigt wird. Mit diesem Befehl können alle Zeichen der Tastatur – manchmal mit Ausnahme der Anführungsstriche – eingegeben werden. Für die Anzeige der Tastatureingabe muß ein besonderer PRINT-Befehl geschrieben werden – besonders bei Verwendung von INKEY$.

Vielfach können aber Zahlen nur einer String-Variablen zugewiesen werden.

Der schwerwiegendste Nachteil bei der Eingabe mit GET oder INKEY$ liegt bei den meisten BASIC-Versionen in der mangelnden Korrekturmöglichkeit. Die Löschtaste wirkt zwar auf den Bildschirm, setzt aber (wenn eine Variable aus den eingegebenen Zeichen zusammengestellt wird) auch den Tastencode für die Löschtaste ein. Die Variable wird dann zwar richtig angezeigt, kann aber in einem Vergleichsbefehl nicht erkannt werden. Wie dieser Nachteil zu vermeiden ist, wird im Abschnitt 6.1 erklärt.

LINE INPUT

Die Nachteile beider Befehle vermeidet der bei vielen neueren BASIC-Versionen vorhandene LINE INPUT-Befehl. Er ermöglicht, wie GET und INKEY$, die Eingabe aller Tastaturzeichen, versetzt den Computer aber auch in Wartestellung. Die Übernahme der Eingabe in eine Variable erfolgt erst nach RETURN, so daß die Löschtaste für Korrekturen wirksam bleibt.

4.6 Bildschirm-Ausgabe

```
10 REM  TEST 22 Komma und Semikolon
100 FOR J = 1 TO 5
110 FOR K = 0 TO 9
120 PRINT K; : NEXT K
130 PRINT : NEXT J
140 PRINT 1;2;-3;-4
150 PRINT 7,8,9
160 A$="BASIC"
170 PRINT A$;A$,A$
RUN
 0  1  2  3  4  5  6  7  8  9
 0  1  2  3  4  5  6  7  8  9
 0  1  2  3  4  5  6  7  8  9
 0  1  2  3  4  5  6  7  8  9
 0  1  2  3  4  5  6  7  8  9
 1  2 -3 -4
 7          8          9
BASICBASIC  BASIC
READY
```

TEST 22, Komma und Semikolon
Wie wirken Trennzeichen im PRINT-Befehl? Zahl der Leerstellen vor und hinter Zahlen und Zeichen nach dem Semikolon und Abstand bei den Kolonnen nach dem Komma differieren.

Ausgaben dienen unter anderem zur Darstellung der Ergebnisse der Computerarbeit und zum Dialog mit dem Computer. Während es in allen BASIC-Dialekten praktisch nur einen Ausgabebefehl gibt, sind die Möglichkeiten der Positionierung, also der Programmierung jener Stelle auf dem Bildschirm, an der die Ausgabe erscheinen soll, recht vielfältig.

Der PRINT-Befehl

Der PRINT-Befehl läuft in allen BASIC-Versionen im wesentlichen gleich. Dem Befehlswort können Variable, Rechenausdrücke oder in Anführungsstrichen auch Zeichenketten folgen. Geringe Unterschiede gibt es lediglich bei Verwendung der erlaubten Trennzeichen, Komma und Semikolon.

Für die Ausgabe verfügbare Zeichen

ASCII Code

hex	dez.	ASCII	hex	dez.	ASCII	hex	dez.	ASCII	hex	dez.	ASCII
00	0	NUL	20	32	Space	40	64	@	60	96	\
01	1	SOH	21	33	!	41	65	A	61	97	a
02	2	STX	22	34	"	42	66	B	62	98	b
03	3	ETX	23	35	#	43	67	C	63	99	c
04	4	EOT	24	36	$	44	68	D	64	100	d
05	5	ENQ	25	37	%	45	69	E	65	101	e
06	6	ACK	26	38	&	46	70	F	66	102	f
07	7	BEL	27	39	'	47	71	G	67	103	g
08	8	BS	28	40	(	48	72	H	68	104	h
09	9	HT	29	41	)	49	73	I	69	105	i
0A	10	LF	2A	42	*	4A	74	J	6A	106	j
0B	11	VT	2B	43	+	4B	75	K	6B	107	k
0C	12	FF	2C	44	,	4C	76	L	6C	108	l
0D	13	CR	2D	45	—	4D	77	M	6D	109	m
0E	14	SO	2E	46	.	4E	78	N	6C	110	n
0F	15	SI	2F	47	/	4F	79	O	6F	111	o
10	16	DLE	30	48	0	50	80	P	70	112	p
11	17	DC1	31	49	1	51	81	Q	71	113	q
12	18	DC2	32	50	2	52	82	R	72	114	r
13	19	DC3	33	51	3	53	83	S	73	115	s
14	20	DC4	34	52	4	54	84	T	74	116	t
15	21	NAK	35	53	5	55	85	U	75	117	u
16	22	SYN	36	54	6	56	86	V	76	118	v
17	23	ETB	37	55	7	57	87	W	77	119	w
18	24	CAN	38	56	8	58	88	X	78	120	x
19	25	EM	39	57	9	59	89	Y	79	121	y
1A	26	SUB	3A	58	:	5A	90	Z	7A	122	z
1B	27	ESC	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	'	7C	124	'
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	↑	7E	126	~
1F	31	US	3F	63	?	5F	95	—	7F	127	DEL

Abb. 10: Der ASCII-Code in der internationalen Version
In der deutschen Version sind die Codierungen mit den Dezimalwerten 91, 92, und 93 den großen Umlauten Ä, Ö und Ü zugewiesen. Die kleinen Umlaute haben den Code 123, 124 und 125, das ß 126. Die Bedeutung der Steuerzeichen 0 bis 31 ist nicht ganz einheitlich

Bei allen Computern können die Ziffern und Großbuchstaben sowie eine Reihe von Satz- und Sonderzeichen auf dem Bildschirm gezeigt werden. In der Hauptsache wird der Zeichensatz verwendet, der als ASCII-Code genormt wurde (Abb. 10). In Kleinigkeiten gibt es aber viele Abweichungen, die teilweise durch Eigenheiten der nationalen Schreibweisen bedingt sind.

Da der ASCII-Code nur 7 Bit benutzt, können dort 128 Zeichen untergebracht werden. Hiervon sind die ersten 32 nicht druckbare Steuerzeichen, und nur die Zeichen mit den dezimalen Codewerten 32 bis 127 ergeben eine Anzeige auf dem Bildschirm. In vielen BASIC-Versionen werden aber für den Zeichencode alle 8 zur Verfügung stehenden Bit verwendet, was dann die Darstellung von bis zu 256 Zeichen ermöglicht. Hierbei weisen die verschiedenen Dialekte erhebliche Unterschiede auf.

Die darstellbaren Zeichen können leicht mit der CHR$-Funktion ermittelt werden.

```
10 REM TEST 23 CHR$-Funktion
100 FOR K = 32 TO 126
110 PRINT TAB(POS(1)+2);CHR$(K);
120 IF POS(1) > 40 THEN PRINT
130 NEXT

RUN
       !   "   #   $   %   &   '   (   )   *   +   .   -
   .   /   0   1   2   3   4   5   6   7   8   9   :   :
   <   =   >   ?   §   A   B   C   D   E   F   G   H   I
   J   K   L   M   N   O   P   Q   R   S   T   U   V   W
   X   Y   Z   Ä   ö   Ü   ^   _   `   a   b   c   d   e
   f   g   h   i   i   k   l   m   n   o   p   q   r   s
   t   u   v   w   x   y   z   ä   ö   ü   ß
READY
```

TEST 23, CHR$-Funktion
Das Testprogramm ist auf den Standard-ASCII-Code ausgerichtet. Gegebenenfalls müssen die Anfangs- und Endwerte für die Laufvariable in Zeile 100 geändert werden. In der Zeile 120 wird nur aus drucktechnischen Gründen die Länge der Ausgabezeilen beschränkt. Dies kann auch entfallen.

Bildschirmcode

Einige BASIC-Versionen (z. B. Commodore) benutzen für die Speicherung der Bildschirmdaten einen besonderen Bildschirmcode, der von dem normalen Ausgabecode abweicht. Man kann dann dort alle 256 Codes für darstellbare Zeichen nutzen. Da der Computer intern den allgemeinen Speichercode in den Bildschirm-

code umsetzt, erscheinen eingegebene Steuerzeichen auch als besondere Zeichen auf dem Bildschirm.

Der Bildschirmcode muß verwendet werden, wenn einzelne Zeichen mit dem POKE-Befehl direkt in den Bildschirmspeicher geschrieben werden sollen.

```
10 REM TEST 24 Bildschirmcode
20 CLS: FOR K = 0 TO 126
30 IF K<>0 AND K/32=INT(K/32) THEN Z=Z+1
40 POKE 15680+K+Z*32,K
50 NEXT
90 STOP
100 FOR K = 128 TO 255
120 IF K/32 = INT(K/32) THEN Z=Z+1
130 POKE 15680+K+Z*32,K
140 NEXT

Break in 90
READY
>CONT
READY

§ABCDEFGHIJKLMNOPQRSTUVWXYZÄöü^
 !"#$%&'()*+,-./0123456789:;<=>?
§ABCDEFGHIJKLMNOPQRSTUVWXYZÄöü^_
'abcdefghijklmnopqrstuvwxvzäöüß
```

TEST 24, Bildschirmcode
Das Testprogramm muß an den verwendeten Computer angepaßt werden. In den POKE-Befehl gehört jeweils die erste Adresse des Bildschirmspeichers. Auch die Anzahl der druckbaren Zeichen differiert in den verschiedenen BASIC-Dialekten.
Die Zeichen werden im Testprogramm in zwei getrennten Schleifen auf den Bildschirm geschrieben. Da der Code 127 bei der verwendeten Computer/Drucker-Kombination als Löschbefehl für den Pufferspeicher im Drucker wirkt, darf dieser Code nicht in einer zu druckenden Ausgabe vorkommen. Die Zeilen 30 und 120 bewirken nur eine Verkürzung der Zeilenlänge. Die Zeichen ab Code 128 können auf dem verwendeten Drucker nicht in der Hardcopy gedruckt werden.

Positionierung der Ausgaben

Normalerweise beginnt die Ausgabe auf dem Bildschirm nach einem PRINT-Befehl immer am Anfang der nächsten Zeile. Mit der TAB-Funktion kann der Beginn der Ausgabe auf die in der Klammer folgende Tabulatorstelle gelegt werden. Bei dieser Funktion ist zu beachten, daß meist kein Leerzeichen vor der Klammer stehen darf. Zwischenräume können mit der SPC-Funktion oder

einfach durch Leerzeichen in einer Zeichenkette programmiert werden.

Soweit gibt es wenig Unterschiede bei den BASIC-Dialekten. Unterschiedlich sind aber schon die Befehle, die zu geben sind, wenn der Bildschirm gelöscht und der Cursor auf die Anfangsposition gesetzt werden soll. Bei einigen Computern heißt der Befehl hierfür **CLS**, bei anderen **HOME**; und es gibt daneben Computer, bei denen das Bildschirmlöschen und das Rückführen des Cursors getrennt durch Tastenbetätigung programmiert werden muß. Auch der Anruf eines Unterprogramms zu diesem Zweck ist üblich.

In jedem Fall beginnt die neue Ausgabe in der Position, in der der Cursor vor dem PRINT-Befehl steht. Für die Positionierung des Cursors sind im wesentlichen sechs verschiedene Methoden gebräuchlich.

V- und H-Tabulator

Einige Computer (z. B. Apple) gestatten die Positionierung des Cursors über getrennte vertikale **(VTAB)** und horizontale **(HTAB)** Tabellatorbefehle. Manchmal muß danach die absolute Position, manchmal aber auch der relative Abstand zur letzten Position eingesetzt werden.

Cursorpositionierung durch CHR$-Funktionen

Meist gibt es – leider nicht einheitliche – ASCII-Codewerte für die Steuerzeichen zur Cursorbewegung. Sie können dann nach PRINT-Befehlen mit der CHR$-Funktion aufgerufen und in dieser Form auch in das Programm geschrieben werden.

Cursorpositionierung mittels Pfeiltasten

Einige Computer (z. B. Commodore und MZ 80) erlauben durch Verwendung von Pfeiltasten Cursorbewegungen auch innerhalb eines Programms. Die Anweisungen müssen dann innerhalb von Zeichenketten nach einem PRINT-Befehl stehen. In einer Programmliste erscheinen sie manchmal als charakteristische Negativzeichen, die für den Unkundigen nur schwer zu deuten sind.

Positionierung mit PRINT@

Recht einfach ist die Positionierung bei denjenigen Computern (z. B. Genie, TRS 80), bei denen die Bildschirmpositionen durchgehend numeriert sind. Mit Hilfe des PRINT@-Befehls kann die Ausgabe an die Bildschirmposition gesetzt werden, die dem

Befehlswort PRINT@ folgt. Hierbei kann der Computer die Position auch aus einem dort stehenden Rechenausdruck berechnen. Nach der Positionsnummer muß ein Komma stehen, dem dann die zu druckenden Informationen folgen. Abgeschlossen wird der Befehl am besten mit einem Semikolon, um den weiteren Zeilenvorschub zu verhindern. Mit nur einem Semikolon nach dem der Zeilennummer folgenden Komma kann auch die Eingabe auf einen INPUT-Befehl hin an eine bestimmte Bildschirmstelle gesetzt werden.

Positionierung mit LOCATE oder PRINT AT

Beide Befehle arbeiten mit Zeilen- und Spaltennummern, die, durch ein Komma getrennt, dem Befehlswort folgen müssen. Im übrigen entspricht die Anwendung dem PRINT@-Befehl.

Positionierung über POKE-Befehle

Da jede Position des Bildschirmspeichers eine feste Adresse hat, kann man in diese die Zeichencode direkt mit POKE-Befehlen einschreiben. Diese Methode erlaubt einen schnellen Wechsel der Anzeige, erfordert aber auch erheblichen Programmieraufwand.

Die Erzeugung eines gut proportionierten Bildes auf dem Schirm kann also sehr unterschiedlich programmiert werden. Es ist darum meist nicht möglich, einen Programmteil mit positionierter Ausgabe so ohne weiteres zu übernehmen, wenn er auf einem anderen Computer entwickelt wurde.

4.7 Vergleichsoperationen

Die zugelassenen Vergleichsoperatoren sind in allen BASIC-Versionen gleich: Unterschiede gibt es aber in der Wirkung und bei den Anwendungen.

Meldung des Vergleichsergebnisses

Trifft der Computer im Programmlauf auf einen Vergleichsausdruck, stellt er fest, ob dieser »wahr«, also richtig formuliert, oder »falsch« ist. Je nach dem Ergebnis wird intern eine Zahl gesetzt, die man auch *Merker* nennt. Sie kann meist abgefragt oder ausgegeben werden. Es gibt allerdings Unterschiede, welchen Wert dieser Merker bei »wahr« und »falsch« hat.

Vielfach kann mit den Zahlen im Vergleichsmerker gerechnet werden. Man kann dann mehrere in Klammern geschriebene Vergleiche durch Rechenoperatoren verbinden und das Ergebnis abfragen.

```
10 REM TEST 25 Vergleichsergebnisse
20 A = 4 : B = 5
30 PRINT"Ausgabe bei WAHR => ";A < B
40 PRINT"Ausgabe bei FALSCH => ";A > B
50 GOTO 110
100 PRINT"Keine Ausgabe Vergleichsergebnis"
110 C = (A>B)+(B>A)+(A=B)+NOT(A=B)
120 PRINT"2 Vergleiche falsch also C=";C
130 END
200 PRINT"Kein Rechnen mit Vergleichsergebn.

RUN
Ausgabe bei WAHR => -1
Ausgabe bei FALSCH =>  0
2 Vergleiche falsch also C=-2
READY
>
```

TEST 25, Vergleichsergebnisse
Das Testprogramm ermittelt zuerst, welche Merker bei wahrem und falschem Vergleichsergebnis erzeugt werden, und prüft dann, ob mit diesen Zahlen auch gerechnet werden kann.

Bei einigen BASIC-Versionen sind verkürzte Vergleiche nach IF zulässig. Dies bedeutet, daß eine dort stehende Zahl (oder Zahlenvariable) als Vergleichsmerker interpretiert wird.

```
10 REM TEST 26 Verkürzter Vergleich
20 INPUT A
30 IF A THEN PRINT"RICHTIG" : GOTO 20
40 PRINT"FALSCH" : GOTO 20

RUN
? 1
RICHTIG
? -1
RICHTIG
? 0
FALSCH
? 123
RICHTIG
? 123456
RICHTIG
```

TEST 26, Verkürzter Vergleich
Das Programm erlaubt es auszuprobieren, welche Zahlen als falsches oder richtiges Vergleichsergebnis interpretiert werden.

Verkettete Vergleiche

Unterschiedlich ist weiterhin das Laufverhalten, wenn in einem Befehl mehrere Vergleiche zusammengefaßt werden. Man kann hierfür die logischen Operatoren **AND** und **OR** benutzen. Oft ist dies aber auch durch Verschachtelung mehrerer IF-Abfragen vor dem THEN möglich.

```
10 REM TEST 27 Verschiedene Vergleiche
20 A=5: B=6: C=7: D$="5"
30 PRINT"A=5: B=6: C=7: D$=":CHR$(34);"5"
40 IF A < B THEN PRINT"A < B ist wahr
50 IF A<B IF B<C THEN PRINT"A<B und B<C sind wahr
60 IF(A<B)+(B<C)=-2 THEN PRINT"A<B und B<C ist wahr
70 IF(A>B)+(B<C)=-1 THEN PRINT"A>B oder A<B ist wahr
80 IF VAL(D$)=A THEN PRINT"D$ ist gleich A
90 IF D$=CHR$(53) THEN PRINT"CHR$(53) ist 5
100 IF ASC(D$)=53 THEN PRINT"ASCII Code von D$=53
110 IF A=VAL(CHR$(53)) THEN PRINT "A ist gleich D$
120 IF STR$(A)=D$ THEN PRINT"A als String ist = D$:END
130 IF MID$(STR$(A),2,1)=D$ THEN PRINT
"Die erste Stelle muß abgetrennt
 werden damit Zahl = String ist

RUN
A=5: B=6: C=7: D$="5
A < B ist wahr
A<B und B<C sind wahr
A<B und B<C ist wahr
A>B oder A<B ist wahr
D$ ist gleich A
CHR$(53) ist 5
ASCII Code von D$=53
A ist gleich D$
Die erste Stelle muß abgetrennt
 werden damit Zahl = String ist
READY
```

TEST 27, Verschiedene Vergleiche

Das Programm prüft, ob unterschiedlich formulierte Vergleiche von dem benutzten BASIC-Dialekt angenommen werden, und meldet, wenn dies der Fall ist. Schwierigkeiten können auftreten beim Vergleich einer als Zeichenkette gespeicherten Zahl mit einer durch die STR$-Funktion in eine Kette gewandelten Zahl, wenn dort das (nicht sichtbare) Vorzeichen als erstes Zeichen gespeichert wird.

Einzelbitprüfung

Nicht alle BASIC-Versionen prüfen einzelne Bit in einem Bitmuster. Voraussetzung hierfür wäre, daß bei logischen Operationen die einzelnen Bit eines Byte einzeln verarbeitet werden.

```
10 REM TEST 28 Einzelbitprüfung
20 PRINT: INPUT"Welches Bit prüfen (0..7)";N
30 INPUT"Dez.Zahl eingeben";A%
50 IF (A% AND 2ÄN)=2ÄN THEN PRINT
"Das Bit Nr."N" ist 1": GOTO 20
60 PRINT"Das Bit Nr.";N;"ist 0": GOTO 20
RUN

Welches Bit prüfen (0..7)? 1
Dez.Zahl eingeben? 12
Das Bit Nr. 1 ist 0

Welches Bit prüfen (0..7)? 0
Dez.Zahl eingeben? 127
Das Bit Nr. 0  ist 1

Welches Bit prüfen (0..7)? 0
Dez.Zahl eingeben? 128
Das Bit Nr. 0 ist 0
```

TEST 28, Einzelbitprüfungen
Das Programm prüft, ob ein einzelnes Bit in einem Bitmuster den Wert 1 oder 0 hat. Nur wenn das Programm bei allen Prüfzahlen richtige Ergebnisse ausgibt, unterstützt das verwendete BASIC auch die Einzelbitverarbeitung.*

Alphabetische Ordnung von Zeichenketten

Buchstaben können auf der Basis ihrer ASCII-Codewerte alphabetisch geordnet werden, da das A den kleinsten und das Z den größten Codewert hat. Die meisten BASIC-Dialekte können darum auch eindeutig feststellen, ob ein Wort im Alphabet vor

```
10 REM TEST 29 Alphavergleich
20 A$="HAUS": B$="HANS"
30 IF B$ = A$ THEN 50
40 IF B$ < A$ THEN 60
50 PRINT"Alphavergleich nur 1.Buchstabe":END
60 PRINT"HANS im Alphabet vor HAUS"
RUN
HANS im Alphabet vor HAUS
READY
```

TEST 29, Alphavergleich
Das Programm stellt fest, ob ganze Wörter nach dem Alphabet geordnet werden können, oder ob beim Vergleich von Zeichenketten mit den Operatoren < oder > nur das erste Zeichen berücksichtigt wird.

*Da der in den Beispielen dieses Buches verwendete Drucker mit deutschem Zeichensatz arbeitet, wird statt des Potenzierungsoperators ein Ä und statt @ ein § gedruckt.

einem anderen einzuordnen ist. Voraussetzung hierfür ist, daß alle Buchstaben der Wörter in den Vergleich miteinbezogen werden. Außerdem können natürlich nur Großbuchstaben mit Großbuchstaben und Kleinbuchstaben mit Kleinbuchstaben verglichen werden.

4.8 Direkter Speicherzugriff

POKE- und PEEK-Befehle

Alle voll ausgebauten BASIC-Dialekte kennen Befehle, mit denen Bitmuster direkt in eine Speicherstelle geschrieben bzw. von dort gelesen werden können. Während POKE als Anweisung zu benut-

```
10 REM TEST 30 Speicherzugriff
20 FOR K = 32765 TO 32770
40 PRINT K.PEEK(K)
50 NEXT
60 PRINT"Adresse immer dezimal": END
100 PRINT"Adresse nur bis";K-1;"verwendbar.
darüber Komplementzahl benutzen"
110 FOR K = 32765 TO 32770
120 A = K + 65336 * (K > 32767)
130 PRINT A,PEEK(A)
140 NEXT

RUN
 32765           255
 32766           255
 32767           255
 32768
Ueberlauf  in 40
READY

GOTO 100
Adresse nur bis 32767 verwendbar.
darüber Komplementzahl benutzen
 32765           255
 32766           255
 32767           255
-32568           136
-32567           0
-32566           136
READY
>
```

TEST 30, Speicherzugriff

Mit diesem Test kann geprüft werden, in welcher Form bei PEEK- und POKE-Befehlen die Adresse im Befehl zu schreiben ist. Wenn die Komplementzahl eingesetzt werden muß, wird diese in Zeile 120 berechnet. Wenn die Adresse größer als 32767 ist, wird der Klammerausdruck −1, und es wird −65336 + Adresse gerechnet. Ist die Adresse kleiner als 32767, hat die Klammer den Wert 0, und die Adresse bleibt unverändert.

zen ist, der die Speicheradresse und das zu speichernde Bitmuster als Dezimalzahl folgen müssen, ist PEEK mit nachfolgender Adresse in Klammern als Funktion zu verwenden.

Ein wesentlicher Unterschied besteht jedoch bezüglich der Einsetzung der Adresse bei den BASIC-Dialekten. Am einfachsten für den Benutzer ist es, wenn der ganze Bereich der verfügbaren Adressen als Dezimalzahl eingesetzt werden kann. Manchmal kann es aber dann noch die Beschränkung geben, daß nur RAM-Adressen abfragbar sind.

Andere BASIC-Versionen verlangen, daß Adressen über 32767 als Komplementzahl (Adreßzahl – 65336) eingesetzt werden.

Die VARPTR-Funktion

Einige neuere BASIC-Versionen kennen die VARPTR-Funktion, mit der die Adresse bestimmt werden kann, an der eine bestimmte Variable gespeichert ist. Die Wirkung dieser Funktion fällt aber, je nach der Art der Variablenspeicherung, unterschiedlich aus. Vielfach ermittelt die Funktion nicht die erste Speicheradresse der Variablen (in der oft das erste Zeichen des Namens steht), sondern den Anfang der Datenspeicherung. Bei Zeichenketten kann mit VARPTR nur der Zeiger gefunden werden, der auf den Anfang des eigentlichen Zeichenspeichers zeigt (Abb. 11).

```
10 REM Demonstration Variablenspeicherung
20 A%=100 : B!=100000 : C$="MAIER"
30 PRINT VARPTR(A%),
40 FOR K=-3 TO 1:PRINT PEEK(VARPTR(A%)+K);:NEXT
50 PRINT:PRINT:PRINT VARPTR(B!),
60 FOR K=-3 TO 3:PRINT PEEK(VARPTR(B!)+K)::NEXT
70 PRINT:PRINT:PRINT VARPTR(C$),
80 FOR K =-3 TO 2:PRINT PEEK(VARPTR(C$)+K);:NEXT
90 PRINT : PRINT
100 X=PEEK(VARPTR(C$)+1) : Y=PEEK(VARPTR(C$)+2)
110 FOR K= 0 TO 4: PRINT PEEK(X+256*Y+K);: NEXT
RUN
 27506           2  0  65  100  0

 27511           4  0  66  0  80  67  145

 27518           3  0  67  5  140  106

 77  65  73  69  82
READY
```

Abb. 11: Untersuchung des Variablenspeichers mit VARPTR
Einige BASIC-Dialekte erlauben die genaue Untersuchung der Speicherplätze, in denen Variable abgelegt sind. Die VARPTR-Funktion stellt immer die Adresse des 4. Byte des Bereichs, in dem die im Argument stehende Variable gespeichert ist, zur Verfügung. Der Inhalt dieser Speicherstellen ist dann mit PEEK abzurufen

Unterprogramme in Maschinensprache

Fast alle BASIC-Versionen haben auch Befehle, mit denen ein Unterprogramm in Maschinensprache angerufen werden kann. Hierfür werden meist die Befehlswörter **USR** oder **CALL** verwendet. Es gibt jedoch große Unterschiede in der Anwendung.

Bei einigen Dialekten können Unterprogramme mit einem Namen aufgerufen werden. Andere verlangen, daß die Startadressen der Unterprogramme vorher mit **DEFUSR** definiert werden. Es gibt auch Versionen, bei denen man die Startadresse vorher mit POKE-Befehlen in bestimmte Adressen einschreiben muß.

Unterschiedlich auch die Möglichkeiten, Parameter in ein Unterprogramm zu übergeben bzw. von dort zu übernehmen.

4.9 FOR...NEXT-Schleifen

```
10 REM TEST 31 Verschachtelung FOR..NEXT
100 FOR K1=1 TO 5
105 GOSUB 210
110 FOR K2=1 TO 5
120 FOR K3=1 TO 5
125 GOSUB 210
130 FOR K4=1 TO 5
135 GOSUB 210
140 FOR K5=1 TO 5
145 GOSUB 210
150 FOR K6=1 TO 5
160 FOR K7=1 TO 5
170 FOR K8=1 TO 5
180 FOR K9= 1TO 5
200 NEXT:NEXT:NEXT:NEXT:NEXT:
210 PRINT K1;K2;K3;K4:K5
220 RETURN

RUN
 1  0  0  0  0
 1  1  1  0  0
 1  1  1  1  0
 1  1  1  1  1
 1  1  1  1  2
 1  1  1  1  3
 1  1  1  1  4
 1  1  1  1  5
 1  1  1  1  6
RETURN ohne GOSUB in 220
READY
```

TEST 31, Verschachtelungen

Da die BASIC-Dialekte bei dieser Prüfung sehr unterschiedlich reagieren, kann hier nur ein Vorschlag gemacht werden, wie man die Fähigkeit zu Verschachtelungen prüfen kann. Wenn das BASIC wie im Test läuft, ist es benutzerfreundlich. Das Programm ist, damit der Ausdruck nicht zu lang wird, unvollständig geschrieben und wird doch in den Schleifen angenommen.

Wenn die Befehle für diese Schleifenart auch in allen BASIC-Dialekten gleich lauten, so kann das Laufverhalten doch recht unterschiedlich ausfallen.

Richtige Verschachtelungen mehrerer Schleifen, bei denen die zuletzt angesprochene Laufvariable zuerst wieder im NEXT-Befehl angezogen werden muß, sind allgemein erlaubt. Uneinheitlich ist allerdings die zugelassene Anzahl der Verschachtelungsebenen. Sie kann davon abhängen, ob gleichzeitig noch Unterprogramme laufen.

Es ist bekannt, daß Einsprünge in FOR... NEXT-Schleifen nicht erlaubt sind und immer zu einer Fehlermeldung führen, wenn im Programm das Wort NEXT ohne vorangegangenes FOR angetroffen wird. Aussprünge aus diesen Schleifen sind aber in vielen BASIC-Dialekten gestattet.

```
10 REM TEST 32 Aussprung aus Schleifen
100 FOR K=10 TO 12
110 PRINT K
120 FOR J=1 TO 5
130 PRINT J;
140 IF J = 4 THEN PRINT: GOTO 120
150 NEXT J
160 NEXT K

RUN
 10
 1  2  3  4
 1  2  3  4
 1  2  3  4
 1  2  3  4
 1  2  3  4
 1  2  3  4
 1  2  3  4
Break in 140
READY
>
```

TEST 32, Aussprung aus Schleifen
Mit diesem Test kann geprüft werden, ob der Aussprung aus einer inneren Schleife zum Programmabbruch führt. Manchmal ist das der Fall. Bei einem »gutmütigen« BASIC läuft das Testprogramm ewig und muß mit BREAK abgebrochen werden.

Wenn im verwendeten BASIC-Dialekt der Aussprung aus einer Schleife nicht erlaubt ist, kann man sich mit einem einfachen Trick helfen, der in Abb. 12 gezeigt wird. Wird zum Beispiel ein größeres Feld durchsucht, kann der Programmlauf beschleunigt werden, wenn die Suchschleife nach dem Finden des Schlüssels abgebrochen

wird. Ist dies in der verwendeten BASIC-Version nicht vorgesehen, wird im Entscheidungsbefehl die Laufvariable einfach auf den Endwert oder einen größeren Wert gesetzt, worauf bei NEXT die Schleife ordnungsgemäß zum Ende kommt.

```
10 REM Suchen in einem Feld
20 DIM A$(500)
30 A$(12)="Humboldt"
100 FOR K = 1 TO 100
110 IF A$(K)="Humboldt" THEN H=K: K=500
120 NEXT K
130 PRINT"Humboldt steht in A$ mit Index";H

RUN
Humboldt steht in A$ mit Index 12
READY
```

Abb. 12: Suchen in einem Feld mit Schleifenabbruch
Wenn in einem BASIC-Dialekt der Aussprung aus einer nicht voll abgelaufenen FOR... NEXT-SCHLEIFE Schwierigkeiten macht, kann man diese umgehen, indem bei erfüllter Aussprungbedingung der Schleifenzähler auf den Endwert oder auf einen noch höheren Wert gesetzt wird. Bei NEXT kommt die Schleife dann ordnungsgemäß zum Ende

4.10 Bedienung von Druckern

Für die Ansteuerung eines Druckers kommen bei den verschiedenen BASIC-Versionen zwei unterschiedliche Verfahren zum Einsatz. Eine Reihe von Computern verwendet in ihrem BASIC-Befehlssatz ein besonderes Befehlswort, wenn die Ausgabe statt auf dem Bildschirm auf dem Drucker erfolgen soll. Andere wiederum betrachten den Drucker als ein Peripheriegerät, das genauso anzusprechen ist, wie die übrigen peripher anschließbaren Geräte.

Druckbefehle

Für die erste Methode wird häufig das Befehlswort **LPRINT** gebraucht. Es kann ohne weitere Vorbereitung in das Programm eingesetzt werden. Lediglich die im Befehl nachfolgenden Ausdrücke werden zum Drucker geleitet und dort abgedruckt. Im Programm folgende normale PRINT-Befehle veranlassen, wie üblich, die Ausgabe auf dem Bildschirm.

Die meisten BASIC-Versionen, die in dieser Weise arbeiten, erlauben durch das Befehlswort **LLIST** den Ausdruck der Programmliste auf einem Drucker.

Bei dem anderen Verfahren muß das Drucken mit dem OPEN-Befehl vorbereitet werden. Wird danach der normale PRINT-Befehl verwendet, erfolgt die Ausgabe an den Drucker, statt an den Bildschirm.

Zeilentransport und Druckpositionierung

Bei der Ausgabe auf den Drucker ist immer zu berücksichtigen, daß die Anzahl der Zeichen in einer Zeile auf dem Bildschirm und beim Drucker unterschiedlich sein kann. Im allgemeinen wird das Steuerkommando, das am Ende einer Bildschirmzeile die Fortschaltung zur nächsten Zeile bewirkt, nicht an den Drucker weitergegeben. Auf der anderen Seite macht der Drucker immer einen Zeilenvorschub, wenn seine Zeilenkapazität erschöpft ist.

Befehle zur Positionierung der Bildschirmausgabe haben für die Druckerausgabe meist keine Wirkung. Eine Ausnahme hiervon sind die Funktionen TAB und SPC, die auch von Druckern berücksichtigt werden. Es gibt aber wiederum besondere Positionierungsmöglichkeiten für die Druckerausgabe mit Hilfe von Steuerzeichen.

Steuerzeichen

Uneinheitlich ist die Behandlung der verschiedenen Steuerzeichen des ASCII-Codes durch die Drucker. Allgemein verstanden und richtig ausgeführt werden die Kommandos für Zeilen und Papiertransport, obwohl es dafür bei einigen Druckern Wahlmöglichkeiten durch Schalter gibt. Bei anderen Steuerzeichen reagieren die Drucker aber unterschiedlich. Insbesondere Befehlsfolgen mit ESCAPE (ASCII-Codewert 27) werden recht abweichend interpretiert. Sie werden bei vielen Druckern zum Umschalten auf Fett- oder Spreizdruck und für viele andere Voreinstellungen verwendet.

Schwierigkeiten können bei der Ausgabe an einen Drucker entstehen, wenn dieser in den Befehlen stehende Steuerzeichen nicht annimmt oder anders ausdruckt, als der Bildschirm anzeigt. Insbesondere beim Ausdruck eines Commodore-Programms auf einen nicht voll an diesen Computer angepaßten Drucker kann es vorkommen, daß die in Zeichenketten stehenden Steuerzeichen überhaupt nicht erkannt und nicht gedruckt werden. Wenn man versucht, ein so gelistetes Programm zum Laufen zu bringen, kann man eine böse Überraschung erleben. Das Programm ist ja nur unvollständig dargestellt.

Standardbefehle im erweiterten BASIC		
Programmanweisungen	CLEAR oder CLR	löscht Variablenspeicher
	CLS oder HOME	löscht Bildschirm
	DEFDBL	definiert Variable als doppelt genau
	DEFINT	definiert Variable als Ganzzahl
	DEFSNG	definiert Variable als einfach genau
	DEFSTR	definiert Variable als Zeichenkette
	DEFFN	Definition einer Funktion
	FN	Aufruf definierter Funktionen
	GET oder INKEY$	Tastaturabfrage ohne Anzeige
	INP	enthält Bitmuster vom Eingangsport
	LINE INPUT	Eingabe aller Zeichen bis CR
	LOCATE oder PRINT@	Positionierung der Ausgabe
	MOD	liefert Divisionsrest
	OUT	gibt Bitmuster an Ausgangsport
	RANDOMIZE	Neusetzen des Zufallszahlgenerators
	PRINT USING	formatierte Ausgabe
	SWAP	Tausch von Variablen
Eingriff in Maschinensprache	CALL oder USR	Aufruf Unterprogramm in Maschinensprache
	PEEK	enthält Bitmuster aus Speicheradresse
	POKE	schreibt Bitmuster in Speicheradresse
	VARPTR	enthält 1 Byte eines Variablenspeichers
Drucker	LPRINT	Ausgabebefehl an Drucker
	LLIST	erstellt Programmliste auf Drucker
Programm speichern	LOAD od. CLOAD	Programm von Kassette laden
	SAVE oder CSAVE	Programm auf Kassette speichern
	VERIFY oder LOAD?	Programmspeicherung prüfen
	MERGE	Programme vereinigen
Funktionen	CDBL	in doppelte Genauigkeit wandeln
	CSNG	in einfache Genauigkeit wandeln
	EOF	meldet Ende einer Datei
	FIX oder CINT	in Ganzzahl wandeln
	INSTR	sucht String in anderem String
	POS	enthält Position des Cursors
	STRING$	gibt mehrfach gleiches Zeichen aus
Fehlerbehandlung	ON ERROR GOTO	Anweisung bei aufgetretenem Fehler
	ERL	enthält fehlerhafte Zeilennummer
	ERR	enthält Fehlercode
	RESUME	Sprung nach Fehlerbehandlung
Programmentwicklung	AUTO	selbsttätiges Zeilennumerieren
	DELETE	Zeilen löschen
	RENUM	neu numerieren
	TRON oder TRACE ON	Trace-Betrieb einschalten
	TROFF oder TRACE OFF	Trace-Betrieb ausschalten

Abb. 13: Die wichtigsten Befehle des erweiterten BASIC

5. Die BASIC-Dialekte verschiedener Computer

Die sehr große Anzahl der BASIC-Dialekte macht einen vollständigen Vergleich fast unmöglich und vor allem unhandhabbar. Es hat darum wenig Sinn, Befehl für Befehl in großen Tabellen gegenüberzustellen, so daß in diesem Kapitel ein anderer Weg gewählt wurde.

Viele der heute gebräuchlichen Dialekte gehen auf Entwicklungen der Firma Microsoft zurück. Sie weisen darum viele Ähnlichkeiten auf, aber auch immer wieder gerätespezifische Abweichungen und Ergänzungen. In erster Linie wird darum in diesem Kapitel erklärt, welche Befehle von der quasi genormten BASIC-Hochsprache abweichen, und welche ergänzenden Befehle es bei den einzelnen Computern gibt.

Die hier besprochenen BASIC-Dialekte sind immer im Zusammenhang mit den Computern zu sehen, auf denen sie laufen, und werden darum in entsprechender Anordnung behandelt. Mit Blick auf den Zweck dieses Buches werden in erster Linie BASIC-Versionen besprochen, die für Home- und Personal-Computer bestimmt sind. BASIC-Versionen der Minicomputer und der Groß-EDV, die oft mit vielen sonst nicht üblichen Ergänzungsbefehlen arbeiten, werden nur am Rande beschrieben.

Um die einzelnen Abschnitte übersichtlich zu halten, erwähnen wir darin nur das besonders, was von dem in den Abb. 1, Seite 11, und 13, Seite 91, dargestellten Standard abweicht. Aus Platzgründen mußte auch eine Beschränkung auf die für das Verstehen fremder Programme wichtigen Befehle vorgenommen werden; denn auf 2 bis 3 Seiten kann nicht der Inhalt eines ganzen Handbuchs gezeigt werden.

Zur Erklärung der Befehlssyntax wird folgende Schreibweise verwendet (es werden jedoch nicht immer alle in den Befehlen möglichen Argumente eingesetzt):

LOAD	=	Großbuchstaben für BASIC-Befehlswörter;
name	=	Kleinbuchstaben für vom Programmierer einzusetzende Werte,,Namen usw.;
(. . .)	=	runde Klammern als Teil von BASIC-Befehlen;
<. . .>	=	eckige Klammern für wahlfrei zu verwendende Befehlsteile;
a–y	=	für beliebige numerische Werte;
z	=	für Zeilennummern;
A$–X$	=	für beliebige Zeichenketten.

Viele Computer kennen noch BASIC-Erweiterungen mit ergänzenden Befehlen. Sie müssen immer zusätzlich geladen werden und können den Lauf und die Charakteristik des ursprünglich auf dem Computer laufenden BASIC wesentlich verändern. Da es hier unzählige Varianten gibt, können sie nicht mitbehandelt werden.

Einige etwas ältere BASIC-Versionen haben nicht für alle dort möglichen Arbeitsschritte eigene BASIC-Befehle, sondern verlangen die direkte Bedienung von Speicherstellen mit POKE und PEEK. Nur einige wichtige der sich hier anbietenden Möglichkeiten können erklärt werden.

5.1 Das ACORN BASIC

Das auf dem BBC-Computer (ACORN B) und auf dem ACORN ELECTRON laufende BASIC wurde in England entwickelt, was rein äußerlich dadurch zum Ausdruck kommt, daß *MISTAKE* statt ERROR gemeldet wird. Im Gegensatz zum älteren BASIC des ACORN ATOM sind die meisten Befehle des heutigen BASIC-Standards vorhanden, wenn auch zum Teil mit anderer Syntax. Darüber hinaus gibt es aber noch viele zusätzliche Befehle, mit denen – ähnlich wie in Pascal – die strukturierte Programmierung unterstützt wird. Der Zugriff auf die Maschinensprache erfolgt in sonst nicht üblicher Weise. Wenn alle Möglichkeiten des ACORN ausgenutzt werden, sind Programme nur schwer konvertierbar.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl und

signifikanter Groß- und Kleinschreibung gestattet. * Integervariable bis 2.147.483.647 zulässig. * A% bis Z% werden als residente Integervariable nicht gelöscht.

Grundbefehle der BASIC-Hochsprache

CONT-Kommando nicht wirksam; sonst keine Einschränkungen. * Befehlswörter dürfen nur in Großbuchstaben geschrieben werden. * Als Sprungziele sind auch Variable und Rechenausdrücke zugelassen. * Ein Hochkomma im PRINT-Befehl bewirkt Zeilenvorschub. * Nach **INPUT"text"** muß ein Komma folgen. * **RND(x)** erzeugt Zufallszahlen zwischen 1 und x. * **LOG(x)** liefert Logarithmus zur Basis 10.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung. * Nicht vorhandene Befehle : LINE INPUT, MOD, SWAP, INP, OUT, VARPTR, FIX, RESUME, EDIT. * Ausgabe auf den Drucker muß mit **VDU 2** umgeschaltet werden. * GET und INKEY$ haben eine andere Bedeutung.

Statt LOCATE (x,y)	→ **TAB(x,y)** für Positionierung.
Statt PRINT USING	→ **.%** für Formatanweisung.
Statt PEEK (x)	→ **?x** für Speicherlesen.
Statt POKE x,y	→ **?x=y** für Speicherschreiben.
Statt LPRINT	→ **Ctrl.B PRINT** für Druckerausgabe.
Statt LLIST	→ **Ctrl.B LIST** für gedruckte Programmliste.

Wichtige zusätzliche BASIC-Befehle

ADVAL	digitalisiert analoges Signal
ASN (w)	liefert Arcussinus von w
CHAIN (dateiname)	Laden und Abarbeiten mehrerer Programme
DEFPROGname (a,b)	definiert Unterprogramm, übergibt a und b
DEG(r)	wandelt Winkel r von Bogenmaß in Grad
DIV	Operator für Ganzzahldivision
ENDPROC	Ende Unterprogramm
EOR	Operator für exclusives Oder
EVAL (A$)	berechnet Rechenausdruck im String A$
a=GET	übergibt Code einer gedrückten Taste an a
A$=GET$	übergibt Zeichen einer gedrückten Taste an A$

INKEY (x)	wartet x/100 s vor Übergabe des Codes
INKEY$(x)	wartet x/100 s vor Übergabe des Zeichens
HIMEMx	bestimmt höchsten Speicherplatz für BASIC
LOCAL x,y,x$	definiert lokale Variable
LN(x)	liefert natürlichen Logarithmus von x
OLD	hebt das Kommando NEW wieder auf
PROCname	ruft Unterprogramm-Namen auf
RAD(g)	wandelt Winkel g von Grad in Bogenmaß
REPEAT...UNTIL	Schleife mit Abbruchbedingung am Ende
TIME	liefert Zeit in 1/100 Sekunden
VDU a,b,c	führt Steuerbefehle aus
VPOS	liefert vertikale Cursorposition.

Bildschirmaufteilung

Mit dem Befehl **MODE** können 7 verschiedene Bildschirmaufteilungen mit 20, 40 und 80 Zeichen und 25 oder 32 Zeilen in 2 bis 16 Farben gewählt werden. Der Befehl **VDU** ist auch zur Definierung von Text- und Grafik-Bildschirmfenstern verwendbar.

Bildschirmcode

32–127 im Textmodus wie ASCII=Code.
Im Teletext- und im Grafikmodus keine Kleinbuchstaben, dafür aber zusätzliche Grafiksymbole.

Grafik- und Farbbefehle

Hochauflösende Grafik mit bis zu 1280mal 1024 Punkten mit dem Koordinatennullpunkt links unten.

MOVE x,y	bewegt Grafik-Cursor auf x,y
DRAW x,y	zeichnet Linie vom Cursor nach x,y
PLOT a,x,y	zeichnet in der Art a nach x,y
COLOUR f	bestimmt Vorder- und Hintergrundfarbe
GCOL a,f,	ändert Farbe nachfolgender Darstellungsbefehle.

32 beliebige Zeichen können als 8-mal-8-Matrix vom Benutzer mit **VDU 23,a,b,c...** definiert werden.

Befehle zur Tonerzeugung

2 beziehungsweise 4 Tonkanäle können entweder in Maschinensprache direkt oder mit folgenden Befehlen angesprochen werden:

SOUND pliste	erzeugt Ton entsprechend Parameterliste
ENVELOP pliste	bestimmt Hüllkurve für den Ton.

Befehle für die Datenspeicherung

Für Kassetten-, Disketten- und Netzwerkspeicherung werden im wesentlichen die gleichen Befehle verwendet. Die Speicherung erfolgt in Blöcken von 256 Byte mit Prüfbyte. Dateinamen 1...10 Buchstaben lang zulässig. Statt einer Kanalnummer ist der Name einer numerischen Variablen in die Befehle einzusetzen.

a=OPENOUT (name)	öffnet Datei zum Schreiben
a=OPENIN (name)	öffnet Datei zum Lesen
INPUT #,A$	liest Datenfeld in Variable A$
PRINT #,A$	schreibt Variable A$ in Datei.

Besonderheiten

Gewöhnungsbedürftiger Zeileneditor mit COPY-Taste. * Viele Möglichkeiten für die Benutzung von Routinen der Maschinensprache auch in BASIC. * Netzwerkbetrieb in ECONET vorgesehen. * MODE 7 erlaubt Anschluß an Teletext. * 4 Analogeingänge mitA/D-Wandlung ansprechbar.

5.2 Das Applesoft BASIC

Der BASIC-Dialekt, der auf den verschiedenen Apple-Modellen (II, II+, IIe) läuft, ist schon einige Jahre alt und enthält darum nicht alle heute üblichen Befehle. Er ist aber benutzerfreundlich und für alle Aufgaben brauchbar. Als erste weit verbreitete BASIC-Version enthält Applesoft auch schon Befehle für hochauflösende Grafik und Farbdarstellung.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl, aber nur 2 Zeichen signifikant.

Grundbefehle der BASIC-Hochsprache

Nicht vorhanden: ELSE. * Für »wahr« wird +1, für »falsch« 0 erzeugt. * Keine Einzelbitverarbeitung in BASIC. * Positive Zahlen werden ohne Leerstelle ausgegeben und gespeichert.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung. * Nicht vorhandene Befehle : INP, LINE INPUT, MOD, OUT, PRINT USING, SWAP, VARPTR, LPRINT, LLIST, EOF, INSTR

STRING$, AUTO, RENUM. * Zur Ausgabe auf den Drucker muß mit **PRINT "PR #x"** umgeschaltet werden. * Bei GET wartet das Programm unbeschränkt auf eine Eingabe.

Wichtige zusätzliche BASIC-Befehle

FLASH	läßt alle Ausgaben blinken
IN #x	leitet Eingabe auf Slot x
INVERSE	schreibt alle folgenden Ausgaben negativ
NORMAL	folgende Ausgaben wieder normal
HIMEM:Z	bestimmt höchsten Speicherplatz für BASIC
POP	löscht letzte Stückeintragung
PR #x	leitet Ausgabe auf Slot x
SPEEDx	ändert Ausgabegeschwindigkeit
WAIT a,b	warten, bis an Adresse a Bitmuster b ansteht.

Bildschirmaufteilung

In der Grundausstattung werden je 40 Zeichen in 24 Zeilen gezeigt. * Mit Hardwareergänzung sind 80 Zeichen/Zeilen darstellbar. * Mit durch POKE erreichbaren Softwareschaltern können verschiedene Bildmodi gewählt werden. * Ebenso können mit POKE Bildschirmfenster bestimmt werden. * Ohne Erweiterung werden nur Großbuchstaben normal, invers und blinkend dargestellt.

Bildschirmcode

0– 31	Großbuchstaben, negativ
32– 63	Satz- und Sonderzeichen und Ziffern, negativ
64– 95	Großbuchstaben, blinkend
96–127	Satz- und Sonderzeichen und Ziffern, blinkend
128–159	Kontrollcode, keine Anzeige
160–191	Satz- und Sonderzeichen und Ziffern, normal
192–223	Großbuchstaben, normal
224–255	(Kleinbuchstaben).

Grafik- und Farbbefehle

Niedrigauflösende Grafik wird mit **GR**, Text mit **TEXT** aufgerufen. Der Koordinatennullpunkt ist links unten.

COLOR=f	bestimmt Vorder- und Hintergrundfarbe
PLOT x,y	setzt Punkt in x,y
HLIN a,e AT y	zeichnet horizontale Linie von a nach e in Zeile y
VLIN a,e AT x	zeichnet vertikale Linie von a nach e in Spalte x.

Hochauflösende Grafik mit 192mal 280 Punkten wird mit **HGR** für Bildschirmseite 1 oder **HGR 2** für Bildschirmseite 2 aufgerufen.

HCOLOR=f	bestimmt Farbe
HPLOT x,y	setzt Punkt in x,y
HPLOT x,y TO u,v	zieht Linie von x,y, nach u,v.

Eine mit POKE-Befehlen definierte Figur kann mit **DRAW** abgebildet, mit **ROT** rotiert und mit **SCALE** in der Größe verändert werden.

Befehle für die Datenspeicherung

Numerische Datenfelder können auf eine Kassette mit **STOREx** gespeichert und mit **RECALx** wieder gelesen werden. Als x ist der Feldname einzusetzen.

Diskettenbetrieb wird durch das Apple DOS 3.3 unterstützt. Alle DOS-Befehle müssen dem Befehl **PRINT CHR$ (4)**, der oft auch **D$="": PRINT D$** geschrieben wird, in Anführungszeichen folgen.

OPEN name, Sa, Db, Vc	öffnet Datei mit dem genannten Namen mit Ausgang über Slot a auf Diskette b mit der Diskettennummer c
WRITE name	veranlaßt nachfolgende PRINT-Befehle, in die Datei zu schreiben
READ name	veranlaßt nachfolgende INPUT-Befehle, Daten von der Datei zu holen
APPEND name	erlaubt, Daten an bestehende Datei anzufügen.

Für Dateien mit wahlfreiem Zugriff muß noch der Parameter **Ld** in den Öffnungsbefehl aufgenommen werden, wobei d die maximale Länge der Datensätze in Byte bestimmt. Zum Schreiben und Lesen in diesen Dateien wird den Befehlen **Rn** mit n für die Satznummer beziehungsweise **Bm** mit m für die Bytenummer angehängt.

Besonderheiten

Viele Ausbaumöglichkeiten durch steckbare Hardwarekarten.
Gewöhnungsbedürftiger Editor mit CTRL-Taste zu erreichen. * Viele Routinen der Maschinensprache auch in BASIC zugänglich. * Definition von beweglichen Figuren bei hochauflösender Grafik. * Schneller Wechsel zwischen 2 Bildschirmdarstellungen und Definition von Bildschirmfenstern über POKE mit Softwareschaltern.

5.3 Das BASIC des Alphatronic PC

Auf dem Alphatronic läuft ein modernes voll ausgebautes Microsoft-BASIC, das in erster Linie für geschäftliche Nutzung gedacht ist und darum nur beschränkte Möglichkeiten für Grafik- und Tonerzeugung bietet.

Variable und Variablennamen

Zahlenvariable mit einfach- und doppeltgenauer Rechnung. * Variablennamen in Groß- und Kleinschrift bis zu 40 Stellen signifikant.

Grundbefehle der BASIC-Hochsprache

Funktionswerte werden mit doppelter Genauigkeit gerechnet. * Das Fragezeichen nach INPUT wird unterdrückt, wenn nach dem Text statt eines Semikolons ein Komma gesetzt wird. * RESTORE ist auch mit Zeilennummer zulässig.

Befehle des erweiterten BASIC

Keine Einschränkungen.

Wichtige zusätzliche BASIC-Befehle

DEF USR	Festlegen der Einsprungadresse von Maschinenprogrammen
ERASE	Aufheben von Felddimensionierungen in Zeile z
HEX$(a)	wandelt a in Hexadezimalzahl
KEY x, "Z"	belegt Taste x mit String Z
KEY LIST	erzeugt Belegungsliste der Funktionstasten
MID$	auch zum Einsetzen von Zeichenketten
MON	Aufruf des Monitors für Maschinensprache
OKT$(a)	wandelt a in Oktalzahl
INPUT #n, variablenliste	liest Daten von Dateinummer n in Variable
LINE INPUT #n, X$	liest bis zum CR in Variable X$ ein
OPTION BASE	bestimmt Index des 1. Feldelementes
WAIT p,a	warten, bis am Port p die in a gesetzten Bits = 1 sind.

Bildschirmaufteilung

Je nach Einstellung mit **WIDTH** werden 40 oder 80 Zeichen in 25 Zeilen gezeigt. * Festlegen eines Bildschirmfensters ist mit **CONSOLE** möglich. * Ausgaben im *Rollmodus* (bei vollem Fenster wird eine Zeile nach oben geschoben) oder *Blattmodus* (bei vollem Fenster wird eine Zeile überschrieben). * Blinken und Negativdarstellung mit dem Attribut im Colorbefehl möglich.

Bildschirmcode

0– 31	Grafikzeichen
32–127	ASCII-Code in deutscher Version
128–159	Grafikzeichen
160–223	fremdsprachliche Sonderzeichen
224–255	Grafikzeichen.

Grafik- und Farbbefehle

Niedrigauflösende Grafik mit 6-Pixel-Symbolen wird mit GRAPH-Taste eingeschaltet.

COLOR v,h,a	bestimmt Vorder- und Hintergrundfarbe
PRESET (x,y)	setzt einen Punkt.

Befehle für die Datenspeicherung

Diskettenbetrieb wird ohne ein zusätzliches DOS unterstützt. Sequentielle und wahlfrei lesbare Datenspeicherung ist möglich.

OPEN"1:name" FOR m AS n	öffnet Datei mit dem genannten Namen im Modus m als Nummer n auf dem Laufwerk 1
PRINT #n, variablenliste	schreibt Daten in Dateinummer n
WRITE #n, variablenliste	wie PRINT, aber Trennzeichen selbsttätig gesetzt.

Für Dateien mit wahlfreiem Zugriff muß der Aufbau der Datensätze in einem Befehl vorher festgelegt werden, der einen entsprechenden Pufferspeicher bereitstellt. Es werden nur Zeichenketten gespeichert.

FIELD#n,c AS X$, d AS Y$	Datensatz mit den Feldern X$, maximale Länge c und Y$, Länge d
LSET X$ = A$	schreibt A$ linksbündig in Puffer X$

RSET X$ = A$	schreibt A$ rechtsbündig in Puffer X$
CVI CVS CVD	wandeln Zahlen in Zeichenketten
MKI$ MKS$ MKD$	wandeln Zeichenketten wieder in Zahlen
PUT #n,s.	schreibt Daten vom Puffer n in Satz s
GET #n,s	holt Daten vom Satz s in den Puffer n
LOC(n)	liefert zuletzt bearbeitete Satznummer
LOF(n)	liefert Zahl der Sätze in Datei n.

Besonderheiten

Editierung mit Bildschirm oder Zeileneditor möglich. * Monitor für Maschinensprache und DOS im Grundbefehlssatz vorhanden. * Tastenbelegung nach deutscher Norm. * Schriftzeichen für viele Fremdsprachen verfügbar.

5.4 Das Atari BASIC

Das auf den verschiedenen Atari-Modellen laufende BASIC weicht in vielen Kleinigkeiten vom heute Üblichen ab. Es ist darum nicht ganz einfach, fremde Programme auf einem Atari-Computer zum Laufen zu bringen. Da das Atari-BASIC vor allem für Spielprogramme geschaffen wurde, bietet es viele Möglichkeiten für Grafik- und Tonerzeugung, die allerdings nicht sämtlich allein mit BASIC-Befehlen nutzbar sind.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl. * Maximal sind 128 Variable im Programm zugelassen. * Die Länge von Zeichenkettenvariablen muß dimensioniert werden. * Stringfelder werden nicht akzeptiert.

Befehle der BASIC-Hochsprache

Nicht vorhanden: ELSE, SPC, TAN, LEFTS, RIGHTS. * Für »wahr« wird +1, für »falsch« −1 erzeugt. * Das Fragezeichen statt PRINT wird im Listing nicht übersetzt. * **B$ (5,10)** bildet Teilstring aus B$ vom 5. bis zum 10. Buchstaben. * Vor der Verwendung von GET muß der Kanal zur Tastatur geöffnet werden.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung. * Keine Befehle für Tracebetrieb. * Nicht vorhandene Befehle : INP, LINE INPUT, MOD, OUT, PRINT USING, SWAP, VARPTR, LPRINT, LLIST, EOF, INSTR, POS, STRING$, AUTO, DELETE, RENUM.
Statt CLEAR → **CLR** (löscht auch Dimensionierung)
Statt CLS → **PRINT CHR$ (125)** (Bildschirmlöschen)
Statt LOCATE x,y → **POSITION x,y** (Cursorpositionierung)
Statt ON ERROR GOTO → **TRAP** (Fehlerbehandlung)
Statt LLIST → **LIST"P"** (Druckerausgabe des Programms).

Wichtige zusätzliche BASIC-Befehle

BEY	Abschalten von BASIC
COM	entspricht dem DIM-Befehl
DEG	nachfolgende Winkeleingaben in Grad
DOS	zeigt DOS Menü und startet DOS
LOCATEx,y,a	übergibt Zeichen in x,y an Variable a
POP	löscht letzte Eintragung im Stapelspeicher
RAD	nachfolgende Winkeleingaben in Radian
STATUS #k	meldet Status Kanal k nach letzter Operation
CLOG	liefert dekadischen Logarithmus
PADDLE	liefert Position des Drehreglers
PTRIG	meldet, wenn Taste des Steuerknüppels gedrückt
STICK	liefert Position des Steuerknüppels.

Bildschirmaufteilung

Im Textmodus werden je 40 Zeichen in 24 Zeilen gezeigt. * Mit dem **GRAPHICS**-Befehl lassen sich verschiedene andere Darstellungen wählen. * Ein Bildschirmfenster für Texte ist auch bei einigen Grafikmodi möglich.

Bildschirmcode

Textmodus

0– 31	Grafiksymbole
32–127	wie ASCII-Code, internationale Version
128–255	wie 0–127, negativ.

Im *Grafikmodus* kann mit POKE-Befehlen zwischen 2 Zeichensätzen umgeschaltet werden.

Grafik- und Farbbefehle

Verschiedene Arten der Grafikdarstellung werden mit **GRAPHICS** x angewählt. * Bis 92mal 320 Punkte Auflösung bei 4 aus 255 Farben möglich. * Koordinatennullpunkt links oben.

COLORx	bestimmt Farbregister für nachfolgende Befehle
PLOT x,y	setzt Punkt in x,y
DRAWTO x,y	zeichnet Linie nach x,y
SETCOLOR	bestimmt Farbe und Helligkeit im Farbregister.

Sprites werden durch Verändern einzelner Zeichen im Zeichensatz und Speicherung in einer Displayliste in einem besonderen Microprozessor erzeugt. Programmierung weitgehend in Maschinensprache.

Befehle zur Tonerzeugung

Für 3+1 Tonkanäle kann getrennt mit **SOUND** die Tonhöhe, das Rauschen und die Lautstärke programmiert werden. Die Tondauer ist durch eine Schleife zu bestimmen.

Befehle für die Datenspeicherung

Diskettenbetrieb wird durch das residente BASIC unterstützt. Erfolgt der Datenverkehr nicht in üblicher Weise mit der Tastatur und dem Bildschirm, ist ein Datenkanal zur Diskette zu öffnen.

OPEN,m,O,"g:name"	öffnet Datei mit dem genannten Namen auf dem Kanal n in der Betriebsart m auf dem Gerät g
PRINT #k:"daten"	schreibt Daten auf Datei in Kanal k
INPUT #k: x$	liest Daten von Kanal k in Variable X$
GET #k,b	liest Code eines Byte in Variable b
PUT #k,a	speichert a als Byte.

Dateien mit wahlfreiem Zugriff werden nur bedingt unterstützt, indem der Dateizeiger mit POINT auf ein bestimmtes Byte in einem bestimmten Sektor gesetzt werden kann.

Besonderheiten

Tabelle der verwendeten Variablennamen wird zusammen mit dem Programm abgespeichert. * Bildschirmeditor vorhanden. * Nur Zeilennummern bis 32767 zulässig. * Viele Möglichkeiten für grafische Farbdarstellung und Tonerzeugung, aber komplizierte Syntax.

5.5 Das BASIC des CASIO FP-200

Wie bei anderen Aktentaschencomputern läuft auch auf dem CASIO FP-200 ein modernes, gut ausgebautes BASIC. Zusätzlich können allein oder auch zusammen mit BASIC die Befehle eines Tabellen-Kalkulationsprogramms verwendet werden, um Daten in Form einer Matrix darzustellen, zu verarbeiten oder zu speichern.

Variable und Variablennamen

Zahlenvariable mit einfach- und doppeltgenauer Rechnung. * Variablennamen in Groß- und Kleinschrift bis zu 255 Stellen signifikant.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen. * RESTORE ist auch mit Zeilennummer zulässig.
Statt FRE(x) → **SYSTEM** (freier Speicherraum).

Befehle des erweiterten BASIC

Keine Einschränkungen.

Wichtige zusätzliche BASIC-Befehle

DATE$	enthält Datum
DEF USR	Festlegen der Einsprungadresse von Maschinenprogrammen
FRAC(a)	liefert Bruchteil der Dezimalzahl a
KEY x,"Z"	belegt Taste x mit String Z
KEY LIST	erzeugt Belegungsliste der Funktionstasten
LGT(a)	liefert dekanischen Logarithmus von a
OPTION BASE	bestimmt Index des 1. Feldelementes
TIME$	enthält Zeit.

Außer diesen auch sonst üblichen Zusatzbefehlen bietet der FP-200 die Möglichkeit, auf einfache Weise statistische Auswertungen zu machen.

STAT CLEAR	Aufruf für statistische Rechnungen
STAT x	Eingabe von Werten zur Verarbeitung
CNT	zählt eingegebene Werte

SUM x	liefert Summe der x-Werte
MEAN x	liefert Mittelwert der x-Werte
SD x	liefert Standardabweichung.

Bildschirmaufteilung

8 Zeilen mit je 20 Zeichen und 64mal 160 Punkte Grafik können gemischt erzeugt werden. * Koordinatennullpunkt links oben.

Bildschirmcode

0– 31	Grafikzeichen
32–127	ASCII-Code, internationale Version
128–159	Grafikzeichen
160–223	Leerstellen
224–255	Grafikzeichen.

Grafikbefehle

DRAW x,y	setzt Punkt in x,y
DRAW x,y–u,v	zieht Linie von x,y nach u,v
DRAW–x,y	zeichnet vom letzten Punkt nach x,y
QUAD (x,y)–(u,v)	zeichnet Rechteck mit Eckpunkten x,y, u,v.

Befehle für die Datenspeicherung

Zu speichernde Daten werden mit CETL-Kommandos in eine Tabelle geschrieben, in der auch Sortiervorgänge und Additionen durchgeführt werden können. Die Matrix kann mit einem einfachen Kommando auf eine Kassette oder Diskette geschrieben und von dort wieder gelesen werden.

Arbeitet der Computer in CETL, kann von dort aus ein BASIC-Programm angerufen werden, das den Inhalt einzelner Matrixfelder verarbeitet und das Resultat zurückgibt.

OPEN"1:name" FOR m AS n	öffnet Datei mit dem genannten Namen im Modus m als Nummer n auf dem Laufwerk 1
PRINT #n, variablenliste	schreibt Daten in Dateinummer n
INPUT #n, variablenliste	liest Daten von Dateinummer n in Variable

Für Dateien mit wahlfreiem Zugriff muß der Aufbau der Datensätze in einem Befehl vorher festgelegt werden, durch den ein entsprechender Pufferspeicher bereitgestellt wird. Es werden nur Zeichenketten gespeichert.

FIELD #n,c AS X$, d AS Y$	Datensatz mit den Feldern X$, maximale Länge c und Y$, Länge d
LSET X$ = B$	schreibt B$ linksbündig in Puffer X$
RSET X$ = B$	schreibt B$ rechtsbündig in Puffer X$
CVI CVS CVD	wandeln Zahlen in Zeichenketten
MKI$ MKS$ MKD$	wandeln Zeichenketten wieder in Zahlen
PUT #n,s	schreibt Daten vom Puffer n in Satz s
GET # n, s	holt Daten vom Satz s in den Puffer n
LOC(n)	liefert zuletzt bearbeitete Satznummer
LOF(n)	liefert Zahl der Sätze in Datei n.

Besonderheiten

Bis zu 10 Programme und/oder Tabellen können im Computer gespeichert und bei Bedarf zum Laufen gebracht werden.

5.6 Die BASIC-Dialekte der Commodore Computer

Das schon etwas ältere Commodore-BASIC kennt nicht alle Befehle, die moderne, voll ausgebaute Versionen besitzen. Für viele Programmieraufgaben muß mit POKE und PEEK in den Arbeitsspeicher eingegriffen werden. Da bei den verschiedenen Commodore-Modellen unterschiedliche Adressen gelten, sind deren Dialekte nicht voll kompatibel, auch wenn die BASIC-Befehlswörter im wesentlichen übereinstimmen. Eine große Anzahl von Zusatzmodulen zur Erweiterung des Commodore-BASIC im Angebot.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl, aber nur 2 Zeichen signifikant.

Grundbefehle der BASIC-Hochsprache

Nicht vorhanden: ELSE. Sonst keine Einschränkungen. * Bei älteren Dialekten erfolgt Programmabbruch, wenn nach INPUT nur die RETURN-Taste betätigt wird.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung und zur Fehlerbehandlung, * Nicht vorhandene Befehle : INP, LINE INPUT, MOD, OUT, PRINT USING, SWAP, VARPTR, LPRINT, LLIST, FIX, INSTR, STRING$, AUTO, DELETE, RENUM, TRON, TROFF * Zur Ausgabe auf den Drucker muß der Kanal mit **OPEN 1,4,7** geöffnet werden.
Statt CLS → **PRINT CHR$ (147)** (Bildschirmlöschen)
Statt EOF → Status abfragen (Aus-/Eingabe-Status).

Wichtige zusätzliche BASIC-Befehle

CMD x	leitet Ausgabe auf unter log.Nr. x geöffnetes Gerät
STATUS	enthält Status der letzten Ein-/Ausgabeoperation
TIME	enthält Zeit in 1/60 Sekunden
TIME$	enthält Uhrzeit als Zeichenkette
WAIT x,a	Warten, bis an der Adresse x das Bitmuster a ansteht.

Bildschirmaufteilung

Die verschiedenen Commodore-Modelle haben unterschiedliche Bildschirmaufteilungen. * In jede Position können Buchstaben oder Grafikzeichen gesetzt werden. * Hochauflösende Grafik durch zusätzliche Hard- und Software geboten.

Modell	Zeilen x Zeichen	Adressen Bildschirmspeicher
4000	25×40	32 768–33 767
8000	25×80	32 768–34 767
VC-20	23×22	7 680– 8 185
C-64	25×40	1 024– 2 023

Bildschirmcode

	1. Zeichensatz (Grafik)	2. Zeichensatz (Text)
0– 31	wie ASCII-Code 64–95	wie ASCII-Code 96–127
32– 63	wie ASCII-Code 32–63	wie ASCII-Code 32–63
64– 95	Grafikzeichen	wie ASCII-Code 64–95
96–127	Grafikzeichen	Grafikzeichen
128–255	negativ wie 0–127	negativ wie 0–127

Grafik- und Farbbefehle

Alle Commodore-Modelle erlauben PRINT- und POKE-Grafik mit den Zeichen des Bildschirmcodes.

Der VC-20 und der C-64 bieten Farbdarstellung, wobei die Farbwahl getrennt für Vorder- und Hintergrund durch POKE-Befehle in den Farbspeichern erfolgt. Hierfür werden verwendet:

beim VC-20 die Adressen 38400–38905,
beim C-64 die Adressen 55296–56295.

Mit POKE-Befehlen können Zeichen in einer 8-mal-8-Matrix programmiert und auch in Bewegung dargestellt werden.

Beim C-64 lassen sich mit POKE-Befehlen in einer 21-mal-24-Matrix Sprites definieren. Sie können mehrfarbig erscheinen, bewegt und in der Größe geändert werden.

Befehle für die Tonerzeugung

Mit POKE-Befehlen sind beim C-64 3 Tongeneratoren einzustellen, wodurch mehrstimmige Musikerzeugung möglich ist. Die Programmierung ohne ein unterstützendes Betriebsprogramm ist allerdings schwierig, da es keine Tonbefehle in BASIC gibt.

Befehle für die Datenspeicherung

Periphere Geräte müssen immer mit dem OPEN-Befehl in der Form

OPEN log.dateinr,gernr,sek,adr,"lfwerknr:name,typ,modus"

geöffnet und mit **CLOSE #nr** wieder geschlossen werden.

Diskettenbetrieb wird durch verschiedene DOS-Versionen unterstützt, wobei erst ab Version 4.0 wahlfrei lesbare Dateien aufgebaut werden können. Es gelten folgende Anweisungen:

PRINT #n,x,A$	Variable sequentiell in Datei n speichern
INPUT #n,x,A$	Daten in Datei n lesen und Variable zuweisen
GET #n,A$	liest einzelnes Zeichen von der Datei n
PUT #n, A$	schreibt ein Zeichen von A$ in Datei n
APPEND #n,x	schreibt Variable x an das Ende von Datei n.

Bei der DOS-Version 4.0 wird der etwas einfachere Befehl **DOPEN** verwendet, der es erlaubt, als letzten Parameter hinter L die Satzlänge für eine wahlfrei lesbare Datei vorzugeben.

RECORD #n,s ,<b> ermöglicht nachfolgendem Schreib- oder Lesebefehl, auf Satz s und Byte b zuzugreifen.

Besonderheiten

Viele Ausbaumöglichkeiten durch Hard- und Software-Erweiterungen. * Bildschirmeditor mit gewissen Einschränkungen. * Viele Routinen der Maschinensprache auch in BASIC zugänglich. * Auf der Tastatur eingegebene Steuerbefehle können in einem String im Programm abgespeichert werden. Sie werden dann im Listing als Negativzeichen gedruckt. * Viele Möglichkeiten, die die Hardware der Computer bietet, sind nur durch POKE-Befehle nutzbar.

5.7 Das BASIC des Dragon

Auf dem Dragon läuft ein modernes, recht gut ausgebautes BASIC, das auch sehr wirksam BASIC-Befehle für farbige Grafik und für Tonerzeugung enthält. Es ist allerdings wohl in erster Linie für die Programmierung von Spielen und weniger für kommerzielle Nutzung gedacht.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl, aber nur die beiden ersten Zeichen signifikant.

Befehle der BASIC-Hochsprache

Keine Einschränkungen. * Statt REM auch ein Apostroph zulässig. * RND(a) liefert Zufallszahlen von 1 bis a. * Platz für den Zeichenspeicher muß mit CLEAR reserviert werden.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung. * Nicht vorhandene Befehle : INP, MOD, OUT, SWAP, AUTO.

Wichtige zusätzliche BASIC-Befehle

AUDIO	schaltet Tonausgabe des Recorders ein oder aus
CLOADM "name"	lädt Maschinenprogramm name
DEFUSR n	bestimmt Einsprungadresse für Unterprogramm n
EXEC adr	überträgt Steuerung ab adr an Maschinenprogramm
DEG	nachfolgende Winkeleingaben in Grad
HEX$(a)	liefert a als Hexadezimalzahl
JOYSTK(a)	liefert Stellung des Steuerknüppels
MOTOR	schaltet Motor des Recorders ein oder aus
SKIPF"name"	spult Kassette bis name
TIMER	enthält Stellung der Computeruhr.

Bildschirmaufteilung

Im Textmodus werden je 32 Zeichen in 16 Zeilen gezeigt. * Mit dem SCREEN-Befehl lassen sich 4 andere Darstellungen mit höherer Auflösung (bis zu 192 × 256 Punkte), aber dafür weniger Farben wählen.

Bildschirmcode

0– 31	Grafikzeichen
32–127	wie ASCII-Code, internationale Version
128–255	Grafikzeichen.

Grafik- und Farbbefehle

Neben der PRINT- und POKE-Grafik im Textmodus wird Farbgrafik verschiedener Auflösung mit BASIC-Befehlen unterstützt. * Koordinatennullpunkt links oben.

PCLEAR x	reserviert bis zu 8 Bildschirmseiten für Grafik
PCOPY a TO b	kopiert Bildschirmseite a nach b
COLOR v,h	wählt Vorder- und Hintergrundfarbe
PSET x,y,f	setzt Punkt bei x,y in Farbe f
PRESET x,y	löscht Punkt bei x,y
LINE (x,y,)–(u,v)	zeichnet Linie von x,y nach u,v
DRAW"string"	zeichnet entsprechend der Anweisung in string
CIRCLE (x,y),r	zeichnet Kreis in x,y mit Radius r
PAINT (x,y),f,h	füllt Fläche mit Farbe f bis zur Farbe h.

Der LINE-Befehl erlaubt mit weiteren Parametern auch das Zeichnen von Rechtecken, der CIRCLE-Befehl das Zeichnen von Ellipsen.

Der DRAW-Befehl erlaubt das Zeichnen von Figuren beliebiger Form und Größe mit Hilfe einfacher in einer Zeichenkette gespeicherter Kommandos.

Mit **PUT** können Teile einer Bildschirmdarstellung in einer Matrix gespeichert und mit **GET** wieder aufgerufen werden.

Befehle zur Tonerzeugung

Der Dragon erlaubt die Einblendung von Tonaufzeichnungen auf einer Kassette in BASIC-Programme mit Hilfe der **AUDIO-** und **MOTOR**-Befehle. Darüber hinaus bestehen Befehle zur Erzeugung einfacher Töne und Geräusche.

SOUND h,d	erzeugt Ton der Höhe h und Dauer d
PLAY"string"	wandelt Inhalt einer Zeichenkette in Töne.

Befehle für die Datenspeicherung

Zur sequentiellen Datenspeicherung muß immer ein Kanal geöffnet werden. Der Kanal für den Kassettenrecorder hat die Nummer −1.

OPEN"m", #n, "name"	öffnet Datei mit dem genannten Namen auf dem Kanal n in der Betriebsart m
PRINT #n,a,b,c	schreibt Daten auf Datei in Kanal n
INPUT #n,a,b,c	liest Daten von Kanal n in Variable.

Besonderheiten

Einfach, ohne POKE-Befehle zu programmierende Farbgrafik mit guter Auflösung. * Einblendung von Tonaufzeichnungen auf einer Kassette in ein BASIC-Programm. * Einfache Übernahme einer Grafik als Matrix in ein Feld, das dann wieder an beliebige Stelle des Bildschirms gesetzt werden kann.

5.8 Das BASIC der EPSON Computer

Das auf den verschiedenen EPSON-Computern laufende BASIC ist eine moderne, gut ausgebaute Version. Bis auf einige Spezialbefehle entspricht das BASIC des EPSON Handhelt-Computers

HX-20 dem des Bürocomputers QX-10, der allerdings mit seinem MFBASIC noch viele weitere Möglichkeiten, insbesondere für Grafik, bietet.

Variable und Variablennamen

Zahlenvariable mit einfach- und doppeltgenauer Rechnung. * Variablennamen in Groß- und Kleinschrift bis zu 255 Stellen zulässig, von denen 16 signifikant sind.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen. * RESTORE ist auch mit Zeilennummer zulässig. * Leerstellen vor und nach Befehlswörtern notwendig.

Befehle des erweiterten BASIC

Keine Einschränkungen.

Wichtige zusätzliche BASIC-Befehle

COPY	Ausdruck der Anzeige auf eingebautem Drucker
DATE$	enthält Datum
DAY	enthält Wochentag
DEFUSR x	Festlegen der Einsprungadresse für Maschinenprogramm
ERASE a	Aufheben von Felddimensionierungen
HEX$(a)	wandelt a in Hexadezimalzahl
KEY a,"X"	belegt Taste a mit Zeichenkette X
KEY LIST	erzeugt Belegungsliste der Funktionstasten
LOGIN x	wählt Programm aus dem Menü (HX-20)
OKT$(a)	wandelt a in Oktalzahl
OPTION BASE	bestimmt Index des 1. Feldelementes
STAT	meldet Speicherbedarf (HX-20)
TABCNT	enthält Stand des Bandzählers (HX-20)
TIME$	enthält Zeit
TITLE"n"	gibt geschriebenem Programm den Namen n (HX-20)
WHILE...WEND	Schleifenbefehle
WIDTH x	bestimmt Breite des virtuellen Bildschirms
WIND x	spult Band an Zählerposition x (HX-20)
XOR	Operator für exclusiv oder.

Bildschirmaufteilung

Der HX-20 zeigt in 4 Zeilen je 20 Zeichen und erlaubt Grafik mit

32mal 120 Punkten Auflösung. * Koordinatennullpunkt links oben. * Der QX-10 gibt bis zu 80 Zeichen pro Zeile aus und bietet Grafikdarstellung mit 400 × 640 Punkten.

Bildschirmcode

0–127	ASCII-Code, internationale Version
128–255	Grafikzeichen.

Grafikbefehle des HX-20

PSET (x,y)	setzt Punkt in x,y
PRESET (x,v)	löscht Punkt in x,y
LINE (x,y)–(u,v)	zeichnet (löscht) Linie x,y nach u,v
POINT x,y	prüft, ob Punkt in x,y gesetzt ist
LOCATE$ x	wählt Bildausschnitt.

Befehle für die Datenspeicherung beim HX-20

Durch BASIC wird die sequentielle Datenspeicherung auf dem eingebauten Mikrorecorder oder einem extern angeschlossenen Bandgerät und die wahlfreie lesbare Speicherung in einer RAM-Datei unterstützt. Für die Kassettenspeicherung gelten folgende Befehle:

OPEN"m",#n,"ger:name"	öffnet Datei mit dem genannten Namen im Modus m als Nummer n auf dem mit ger angesprochenen Gerät
PRINT#n,variablenliste	schreibt Daten in Dateinummer n
INPUT#n,variablenliste	liest Daten von Dateinummer n in Variable
CLOSE#n	schließt Datei n.

Für Dateien mit wahlfreiem Zugriff im RAM muß der Aufbau der Datensätze im **DEFFIL**-Befehl vorher festgelegt werden. Ganzzahlvariable benötigen 2 Byte, einfachgenaue 4 Byte und doppeltgenaue Variable 8 Byte. Als letztes kann eine Zeichenkettenvariable im Datenfeld stehen. Die Anfangsadresse jeder Datei muß als relative Adresse im RAM, also bei der ersten Datei mit 0 beginnend, eingesetzt werden. Es gelten folgende Befehle:

DEFFIL l,rel.adr.	definiert Datensätze der Länge 1
PUT% s,variablenliste	schreibt Daten in Satz s
GET% s,variablenliste	holt Daten vom Satz s in Variable
LOF(n)	liefert Zahl der Sätze in Datei n.

Datenspeicherung auf dem QX-10

Die Befehle für die Datenspeicherung beim QX-10 entsprechen dem Standard des Microsoft-BASIC 80, wie beim IBM-PC beschrieben.

Besonderheiten

Datenübertragung vom HX-20 zum QX-10 ist über die serielle Schnittstelle möglich. * Die Flüssigkristallanzeige stellt nur einen Teil des virtuellen Bildschirms dar. * Das MF-BASIC des QX-10 bietet eine Auswahl besonderer, auch vergrößerter, Schriften für Bildschirmdarstellung und Druck.

5.9 Das Hewlett-Packard BASIC der Serie 80

Das auf den älteren H-P-Computern laufende BASIC stimmt in vielen Einzelheiten nicht mit dem heute üblichen Standard überein, hat aber einen großen Befehlssatz, der alle im technischen und wissenschaftlichen Bereich liegenden Aufgaben gut unterstützt. Sehr vielfältig sind auch die Möglichkeiten für die Erstellung von technischen Grafiken nur mit BASIC-Befehlen.

Variable und Variablennamen

Zahlenvariable mit einfach- (SHORT = 5 Stellen) und doppeltgenauer (REAL = 12 Stellen) Rechnung. * Variablennamen in Groß- oder Kleinschrift bis zu 31 Zeichen signifikant. * Variablennamen müssen durch mindestens ein Leerzeichen von numerischen Konstanten und Befehlswörtern getrennt sein.

Grundbefehle der BASIC-Hochsprache

Zeichenketten mit mehr als 10 Zeichen müssen mit der Länge in der ersten Dimension dimensioniert werden. * # und <> als Ungleichoperator. * Mehrfachzuweisung in der Form **a,b,c = 4** zulässig. * Ausrufungszeichen anstelle von REM erlaubt. * Programm muß mit END oder STOP abgeschlossen werden. * RESTORE auch mit Zeilennummer. * Außer DIM auch **REAL, SHORT, INTEGER** zum Dimensionieren von Feldern. * Kein LEFT$ und RIGHT$. * Statt REM auch ! vor Kommentaren.

Statt CLEAR	→ **SCATCH** (Variable löschen)
Statt STOP	→ **PAUSE** (Programm anhalten)
Statt PRINT	→ **DISP** (Bildschirmausgabe)

Statt ASC (X$)	→	**NUM (X$)** (Funktion für Codewert)
Statt STR$(x)	→	**VAL$(x)** (Wandlung Zahl in String)
Statt :	→	\ (Befehlstrennung)
Statt MID$	→	**STR** (Teilstringbildung).

Befehle des erweiterten BASIC

Nicht vorhandene Befehle: MOD, LINE INPUT, SWAP, LLIST, GET, IMP, OUT, PEEK, POKE, VARPTR.

Statt CLS	→	**CLEAR** (Bildschirm löschen)
Statt DEFDBL	→	**REAL** (Definition als doppeltgenau)
Statt DEFINT	→	**INTEGER** (Definition als Ganzzahl)
Statt DEFSNG	→	**SHORT** (Definition als einfachgenau)
Statt LPRINT	→	**PRINT** (Druckerausgabe)
Statt FIX(x)	→	**IP(x)** (Nachkommastellen abtrennen)
Statt INSTR	→	**POS** (Suchen in einem String)
Statt POS	→	**CURSOR** (Stellung des Cursor)
Statt SAVE	→	**STORE** (Programmspeichern)
Statt MERGE	→	**CHAIN** (Programme vereinigen).

Die Positionierung von Ausgaben wird im **IMAGE**-Befehl vorbereitet und erfolgt dann mit PRINT USING.

Wichtige zusätzliche BASIC-Befehle

DATE	enthält Datum
DEG	schaltet Computer in Grad-Modus um
DIV: oder \	Operand für Ganzzahldivision
EXOR	logische exclusiv-oder-Verknüpfung
FP(x)	liefert Dezimalanteil von x
LGT(x)	liefert dekadischen Logarithmus von x
OPTION BASE	bestimmt Index des 1. Feldelementes
PRINT ALL	schaltet alle Ausgaben auf den Drucker
PRINTER IS	bestimmt Ausgabe und Zeilenlänge
RAD	schaltet Computer in Rad-Modus
SET TIME	Anweisung zum Setzen von Zeit und Datum
TIME	enthält Zeit
WAIT x	x Millisekunden warten.

Bildschirmaufteilung

Anzeige von 16 oder 24 Zeilen mit je 80 Zeichen mit **PAGESIZE** wählbar aus einem Speicher mit 54 Zeilen. * Mit **GRAPH** oder **GRAPHALL** Umschaltung auf Grafik-Modus mit dem Nullpunkt links unten.

Bildschirmcode

0–127	ASCII-Code, internationale Version
128–255	wie 0–127, aber negativ.

Grafikbefehle

Das H-P-BASIC enthält komfortable Befehle für grafische Darstellung, insbesondere von Funktionen und Kurven, auf dem Bildschirm oder einem angeschlossenen Plotter. Die meisten Befehle können mit verschiedenen Parametern verwendet werden.

AXIS	zeichnet Achsen und deren Beschriftung
DRAW	zeichnet eine Linie
FRAME	zeichnet einen Rahmen
GRID	zeichnet ein Gitternetz
LIMIT	bestimmt Abbildungsgrenze
LINE	bestimmt Art der zu zeichnenden Linie
LOCATE	bestimmt Zeichengrenze
MOVE	bewegt Stift ohne Zeichnen
PEN	Auswahl des Farbstiftes für Plotter
PDIR	bestimmt Plotrichtung
PLOT	zeichnet eine Linie
SCALE	definiert min und max von x und y.

Befehle für die Datenspeicherung

Datenspeicherung in Datensätzen von 256 Byte mit sequentiellem oder wahlfreiem Zugriff ist programmierbar.

ASSIGN#n TO "name"	öffnet Datei mit dem genannten Namen als Nummer n
PRINT#n:variablenliste	schreibt Daten in Dateinummer n
READ#n: variablenliste	liest Daten von Dateinummer n in Variable
LINE INPUT#n,XS	liest bis zum CR in Variable XS ein.

Bei Dateien mit wahlfreiem Zugriff werden die gleichen Befehle verwendet, nur muß in die Schreib- und Lesebefehle nach der Dateinummer die Satznummer eingesetzt werden.

Besonderheiten

Schrittweiser Lauf der Programme im STEP-Betrieb möglich. * Vielseitige Formatierung mit IMAGE vor PRINT USING. * Möglichkeit zum Abspeichern einer Bildschirmgrafik. * Einbindung von Interrupts in BASIC-Programme möglich. * Einfaches Arbeiten im Direktbetrieb. * Label als Sprungziele zugelassen.

5.10 Das BASIC der IBM Personal-Computer

Das BASIC der IBM Personal-Computer und das der dazu kompatiblen Geräte entspricht weitgehend dem BASIC 80 von Microsoft mit zusätzlichen Befehlen für Datenfernübertragung, Grafik, Farbe und Tonerzeugung. Durch Segmentierung können bis 640 KByte im Hauptspeicher verwaltet werden.

Variable und Variablennamen

Zahlenvariable in einfacher und doppelter Genauigkeit. * Variablennamen in beliebiger Länge mit 40 signifikanten Zeichen.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen * Statt REM auch Apostroph zulässig. * Leerstellen oder Satzzeichen vor und nach Befehlswörtern erforderlich.

Befehle des erweiterten BASIC

Keine Einschränkungen.

Wichtige zusätzliche BASIC-Befehle

CHAIN dateinamen	ruft Programm auf und übergibt Variable
COM(x) ON/OFF/ STOP	startet, beendet, stoppt Datenfernverarbeitung
COMMON var. liste	übergibt Variable an Programm
DATE$	enthält Datum als mm,dd,jjjj Zeichenkette
DEF SEG adresse	definiert Segment des Hauptspeichers
ERASE	löscht Dimensionierung von Feldern
KEY ON/OFF/LIST	zeigt oder listet Inhalt der Funktionstasten
KEY x,X$	setzt Taste x auf Wert X$
KEY(n) ON/OFF	schaltet Funktionstasten ein und aus
LOCATE s,z	setzt Cursor auf Spalte s und Zeile z
MOTOR	schaltet Kassettenrecordermotor
OPTION BASEx	setzt Wert für kleinstes Feldelement
SCREEN	bestimmt Bildschirmanzeige
TIME$	enthält Uhrzeit als hh:mm:ss Zeichenkette
WEND	schließt WHILE-Schleife
WHILE ausdruck	Schleife läuft, solange Ausdruck wahr ist
XOR	Operator für exclusiv oder
&H &O	Vorzeichen für Hex- beziehungsweise Oktalzahlen.

Bildschirmaufteilung

Text Je nach Einstellung mit **WIDTH** in 24 Zeilen mit 40 oder 80 Zeichen. * Zusätzlich eine Statuszeile möglich, die nicht gelöscht wird. * 4 oder 8 Bildschirmseiten können gespeichert werden.

Grafik Mittlere Auflösung 320 × 200 Punkte mit 4 Farben. * Hohe Auflösung 640 × 200 Punkte mit 2 Farben. * Koordinatennullpunkt (1,1) links oben.

Bildschirmcode

0– 31	Steuercode und einige Grafikzeichen
32–127	wie ASCII-Code, internationale Version
128–255	Grafikzeichen.

Grafik- und Farbbefehle

CIRCLE (x,y),r	zeichnet Kreis mit Radius r in x,y
COLOR v,h,r	bestimmt Farben für Vordergrund (v), Hintergrund (h) und Rahmen (r)
DRAW X$	zeichnet Figur entsprechend X$
LINE (x,y)–(u,v)	zeichnet Linie von x,x nach p,q
PAINT (x,y), f,h	füllt ab x,y bis Grenze h mit Farbe f
PRESET (x,v)	setzt Punkt in x,y in Hintergrundfarbe
PSET (x,y)	setzt Punkt in x,y in Vordergrundfarbe
GET (x,y)–(u,v), f	speichert Bildschirmbereich im Feld f
PUT (x,y), f	schreibt die im Feld f gespeicherte Grafik.

Befehle zur Tonerzeugung

BEEP	erzeugt einen kurzen Ton
SOUND f,d	erzeugt Ton der Frequenz f mit der Dauer d
PLAY X$	spielt Musik entsprechend Zeichenkette X$.

Befehle für die Dateiverwaltung

Das Betriebssystem PC-DOS (MS-DOS) unterstützt sequentielle und wahlfrei lesbare Dateien.

Schreibweise des Dateinamens: **"laufwerk:name,erg"**
Laufwerk=A oder B, Name 8 Zeichen, Ergänzung 3 Zeichen.

OPEN"name" FOR OUTPUT AS #n	Schreiben der sequentiellen Dateinummer n

OPEN"0", #n, "name"	dito
OPEN"name" FOR INPUT AS #n	Lesen der sequentiellen Dateinummer n
OPEN"I", #n, "name"	dito
PRINT# PRINT# USING WRITE	Schreiben in Datei
INPUT# LINE INPUT# INPUT$	Lesen aus einer Datei.

Für wahlfrei lesbare Dateien muß die Datensatzlänge im OPEN-Befehl und der Aufbau der Datensätze in einer Feldeinteilung festgelegt werden, wodurch ein Pufferspeicher reserviert wird. Es werden nur Zeichenketten gespeichert.

OPEN"name" AS #n LEN=1	öffnen wahlfreie Datei mit Satzlänge l
OPEN"R", #n, "name",l	dito
FIELD #n, 25 AS Y$, ...	reserviert für Datenfeld Y$ 25 Byte
LSET Y$ = B$	schreibt B$ linksbündig in Feld Y$
RSET Y$ = B$	schreibt B$ rechtsbündig in Feld Y$
MKI$ MKS$ MKD$	wandeln Zahlen in Zeichenketten
CVI CVS CVD	wandeln Zeichenketten wieder in Zahlen
PUT #n,s	schreibt Daten vom Puffer n in Satz s
GET #n, s	holt Daten von Satz s in Puffer n
LOC(n)	liefert zuletzt bearbeitete Satznummer
LOF(n)	liefert Zahl der Sätze in der Datei n
BLOAD BSAVE	Laden und Speichern binärer Dateien.

Besonderheiten

Besondere Befehle für Datenfernverarbeitung vorhanden. * Bildschirmeditor für Programmkorrektur * Es gibt auch noch Befehle zur Arbeit mit einem Lichtgriffel und Steuerknüppeln. * Viele weitere Ausbau- und Erweiterungsmöglichkeiten.

5.11 Die BASIC-Dialekte der LASER Computer

Auf den LASER Computern 310 und 2001 laufen recht unterschiedliche BASIC-Dialekte. Das BASIC des 2001 ermöglicht eine höhere Auflösung und bietet eine Reihe von Grafik- und Tonbefehlen, die der 310 nicht hat. Letzterer beherrscht dafür mehr Befehle des erweiterten BASIC.

Variable und Variablennamen

Zahlenvariable mit 6 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit 5 Zeichen, aber nur 2 Zeichen signifikant.

Grundbefehle der BASIC-Hochsprache

Nicht vorhanden beim 2001: ELSE; sonst keine Einschränkungen. * Nicht vorhanden beim 310: ON...GOTO; sonst keine Einschränkungen. Statt FRE(0) → **RAM(0)−16384**

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung und zur Fehlerbehandlung. * Nicht vorhandene Befehle bei beiden Modellen : DEF FN, LINE INPUT, LOCATE, MOD, SWAP, VARPTR, MERGE, EOF, FIX, INSTR, POS, STRING$, AUTO, DELETE, RENUM. * Beim 310 fehlen außerdem: TRACE, NOTRACE. * Beim 2001 fehlen : INP, OUT, INKEYS, PRINT USING, VERIFY.

Wichtige zusätzliche BASIC-Befehle

COPY	Hardcopy des Bildschirms auf dem Drucker (nur 2001)
VPEEK	Lesen einer Speicherstelle des Videoram (nur 310)
VPOKE	Schreiben in eine Speicherstelle des Videoram (310).

Bildschirmaufteilung

310	16 Zeilen mit je 32 Zeichen
2001	24 Zeilen mit je 40 Zeichen.

Bildschirmcode

310

0− 63	wie ASCII-Code 32−127
64−127	wie 0−63, aber negativ
128−255	4 Pixel-Grafikzeichen in verschiedenen Farben.

2001

0−127	wie ASCII-Code, internationale Version.

Grafik- und Farbbefehle

310 Im Textmodus, aufzurufen mit **MODE (0)**, können 12 Grafikzeichen in vier verschiedenen Farbkombinationen mit POKE in das Bildschirmram (28672−29183) geschrieben werden.

Im Grafikmodus, **MODE (1)**, gelten in einem Raster von 64×128 Punkten folgende Befehle (Koordinatennullpunkt links oben):

SET x,y	setzt Punkt in x,y
RESET x,y	löscht Punkt in x,y
POINT x,y	prüft Punkt in x,y
COLOR v,h	bestimmt Vorder- und Hintergrundfarbe.

2001 Im Textmodus, aufzurufen mit TEXT, können einige Grafikzeichen mit POKE-Befehlen in den Bildschirmspeicher (6400–7167) gesetzt werden. Im Grafikmodus, aufzurufen mit GR, kann in einem Raster von 192 × 256 Punkten mit folgenden Befehlen gezeichnet werden:

PLOT x,y TO u,v	zeichnet Linie von x,y nach u,v
UNPLOT x,y TO u,v	löscht Linie von x,y nach u,v
VPEEK (adr.)	liefert Code, der im Videoram gespeichert ist
VPOKE adr, wert	schreibt Wert in das Videoram
COLOR=v,h	bestimmt Vorder- und Hintergrundfarbe.

Befehle für die Tonerzeugung

310 Auf einem Tonkanal können mit dem Befehl SOUND h,d 31 verschiedene Töne mit unterschiedlicher Dauer programmiert werden.
2001 Auf drei Tonkanälen ist Musikerzeugung möglich. Die Tonkanäle können mit Codeziffern im Befehl **SOUND (h,d,l), ()...** in BASIC programmiert werden. Mit **SGEN** kann direkt auf die Tongeneratoren in Maschinensprache zugegriffen werden.

Befehle für die Datenspeicherung

Der LASER 2001 bietet mit den folgenden Befehlen eine einfache Möglichkeit, Daten auf eine Kassette zu speichern und von dort wieder zu lesen:

STORE a,b,C$, D$	speichert den Inhalt der Variablen
RECALL a,b,C$, D$	liest Daten in Variable ein.

Mit **BSAVE** und **BLOAD** können Daten in Binärform gespeichert und gelesen werden.

Besonderheiten

Bei beiden LASER-Modellen bestehen auch noch Befehle zur Steuerung von Steuerknüppeln. * Überdies gibt es Zusatzmodule für den Diskettenbetrieb mit weiteren Befehlen. * Das BASIC des LASER 210 und des VZ 210 entspricht dem des 310.

5.12 Das LEVEL-II-BASIC des Genie und TRS 80

Das auf den Genie-Computern 1, 2 und 3, dem Colour Genie und den TRS 80-Modellen 1, 3 und 4 laufende BASIC wurde ebenfalls von der Firma Microsoft entwickelt, dann aber zum Teil noch erweitert. Es ist gut ausgebaut und bedienerfreundlich. Farb- und Tonbefehle gibt es nur beim Colour Genie.

Variable und Variablennamen

Zahlenvariable mit einfach- und doppeltgenauer Rechnung. * Variablennamen mit beliebiger Zeichenzahl, aber nur 2 Stellen signifikant.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen. * Apostroph statt REM zulässig. * Mit **CLEAR n** muß Speicherraum für Zeichenketten reserviert werden. **RND** (a) liefert Zufallszahlen von 1 bis a.

Befehle des erweiterten BASIC

Nicht vorhanden : MOD, SWAP. * DEF FN, LINE INPUT, INSTR nur im erweiterten DISK-BASIC. * Sonst keine Einschränkungen. * Für den **PRINT@**-Befehl werden die Plätze des Bildschirmspeichers von 0–1023 beziehungsweise 959 durchgezählt.

Bildschirmaufteilung

Beim Genie 1 und 2 und beim TRS 80-Modell 1 werden, je nach Einstellung, 32 oder 64 Zeichen in 16 Zeilen dargestellt. * Das Colour Genie zeigt 40 Zeichen in 25 Zeilen. * Genie 3 und einige TRS 80-Modelle erlauben auch die Darstellung von 80 Zeichen pro Zeile.

Bildschirmcode

0–127	ASCII-Code
128–191	Grafikzeichen (Zeichensatz 1 beim Colour Genie)
192–255	Grafikzeichen (Zeichensatz 2 beim Colour Genie).

Beim Colour Genie können die Bilder in beiden Grafikzeichensätzen auch frei programmiert werden.

Grafik- und Farbbefehle

Genie 1 und 2, TRS 80/1

Niedrigauflösende Grafik mit 6-Pixel-Symbolen kann mit der Textdarstellung gemischt werden. Es sind 64 × 128 Punkte mit folgenden Befehlen darstellbar (Nullpunkt links oben):

SET x,y	schreibt Punkt nach x,y
RESET x,y	löscht Punkt in x,y
POINT x,y	prüft Punkt in x,y.

Colour Genie

PRINT-Grafik ist mit **LGR** aufzurufen. Es gelten folgende Befehle:

CHAR n	wählt Grafikzeichensatz (n = 0...3)
COLOR v,h	bestimmt Vorder- und Hintergrundfarbe.

Punkt- und Liniengrafik (96 × 160 Punkte) wird mit **FGR** aufgerufen.

PLOT x,y TO u,v	zeichnet Linie von x,y nach u,v
NPLOT x,y TO u,v	löscht Linie von x,y nach u,v
CIRCLE x,y,r	zeichnet Kreis Radius r in x,y
FCOLOUR f	bestimmt Vordergrundfarbe f
FILL f	füllt Bildschirm mit Farbe f
PAINT x,y,f,b	füllt Fläche ab x,y mit Farbe f bis Farbe b.

In einer Figurentabelle können Zeichenanweisungen gespeichert werden, die mit folgenden Befehlen abgerufen werden (siehe auch Abb. 6 auf Seite 46):

SHAPE x,y	zeichnet Figur ab x,y
NSHAPE x,y	löscht Figur ab x,y
SCALE	bestimmt Maßstab für Figur.

Befehle für die Tonerzeugung

Das Colour Genie besitzt 3 Tongeneratoren, die mit den BASIC-Befehlen **PLAY** und **SOUND** und direkt in Maschinensprache angesprochen werden können.

Befehle für die Datenspeicherung

Diskettenbetrieb wird durch verschiedene DOS unterstützt. Sequentielle und wahlfrei lesbare Datenspeicherung ist möglich.

OPEN"m",#n,"name:lfwk"	öffnet Datei mit dem genannten Namen im Modus m als Nummer n auf dem Laufwerk lfwk
PRINT #n,variablenliste	schreibt Daten in Dateinummer n
INPUT #n,variablenliste	liest Daten von Dateinummer n in Variable.

Für Dateien mit wahlfreiem Zugriff muß man den Aufbau der Datensätze in einem Befehl vorher festlegen, durch den ein entsprechender Pufferspeicher bereitgestellt wird. Es werden nur Zeichenketten gespeichert.

FIELD#n, c AS X$,d A$ Y$	Datensatz mit den Feldern X$, maximale Länge c und Y$, Länge d
LSET X$ = A$	schreibt A$ linksbündig in Puffer X$
RSET x$ = A$	schreibt A$ rechtsbündig in Puffer X$
CVI CVS CVD	wandeln Zahlen in Zeichenketten
MKI$ MKS$ MKD$	wandeln Zeichenketten wieder in Zahlen
PUT #n,s	schreibt Daten vom Puffer n in Satz s
GET #n,s	holt Daten vom Satz s in den Puffer n
LOC(n)	liefert zuletzt bearbeitete Satznummer
LOF(n)	liefert Zahl der Sätze in Datei n.

Die auf den genannten Computern laufenden Betriebssysteme NEW-DOS 80 und G-DOS ermöglichen zusätzlich noch den Aufbau von sehr effektiven, wahlfrei lesbaren Dateien mit **IGEL**-Befehlen, in denen mehrere Feldeinteilungen für eine Datei bestimmt werden können. In diesen Dateien können Zahlen auch, wie im Speicher stehend, in verdichteter Form gespeichert werden.

Besonderheiten

Gut verständliches BASIC, da POKE-Befehle, außer für Grafik, unnötig sind. * Bildschirm oder Zeileneditor möglich. * Viele Programmierhilfen im Zusammenhang mit dem DISK-BASIC vorhanden.

5.13 Das BASIC des MSX-Standards

Das MSX-(Microsoft Extended)BASIC ist in erster Linie für Homecomputer gedacht. Es bietet leistungsstarke Grafik-, Farb- und Tonbefehle. Programme, die in diesem BASIC geschrieben wurden, sind auf mehreren Computerfabrikaten lauffähig.

Variable und Variablennamen

Zahlenvariable in einfacher und doppelter Genauigkeit. * Gerechnet wird immer mit 14 Stellen in BCD-Arithmetik, so daß Rundungsfehler wegfallen. * Variablennamen in beliebiger Länge mit 2 signifikanten Zeichen.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen. * Statt REM auch Apostroph zulässig. * RESTORE auch mit Zeilennummer akzeptiert.

Befehle des erweiterten BASIC

Keine Einschränkungen.

Wichtige zusätzliche BASIC-Befehle

BIN$(a)	wandelt a in Binärzahl
CRLIN	liefert vertikale Cursorposition
DEFUSR	bestimmt Einsprungadresse von Maschinenprogrammen
ERASE	löscht Dimensionierung von Feldern
HEX$(a)	wandelt a in Hexadezimalzahl
INPUT$ x	liest String mit x Zeichen von der Tastatur
KEY LIST	zeigt oder listet Inhalt der Funktionstasten
KEY, a X$	setzt Taste a auf Wert X$
MOTOR	schaltet Kassettenrecordermotor
TIME	enthält Zeit seit dem Einschalten in 1/60 s
WAIT p,a	Warten, bis am Port p Bitmuster a anliegt
&B &H &O	Vorzeichen für Bin-, Hex- bzw. Oktalzahlen.

Bildschirmaufteilung

Der **SCREEN**-Befehl erlaubt im 1. Parameter die Wahl verschiedener Darstellungen:

0 = Text mit 24×40 Zeichen
1 = Text mit 24×32 Zeichen
2 = hochauflösende Grafik mit 192×256 Punkten
3 = Vielfarben-Grafik mit 48×64 Punkten.

Bildschirmcode

0–127 wie ASCII-Code, internationale Version
128–255 Grafikzeichen.

Grafik- und Farbbefehle

Bei hochauflösender Grafik gelten folgende Befehle mit dem Koordinatennullpunkt links oben:

LINE x,y–u,v	zeichnet Linie von x,x nach u,v
CIRCLE (x,y),r	zeichnet Kreis mit Radius r in x,y
COLOR v,h,r	bestimmt Farben für Vordergrund (v), Hintergrund (h) und Rahmen (r)
DRAW X$	zeichnet Figur entsprechend X$
PAINT(x,y),f,h	füllt ab x,y bis Grenze h mit Farbe f
PRESET (x,y)	setzt Punkt in x,y in Hintergrundfarbe
PSET (x,y)	setzt Punkt in x,y in Vordergrundfarbe.

Mit dem Befehl **SPRITE$(n) = zeichenkette** können verschiedene Sprites definiert und mit **PUT SPRITE n,x,y** an eine bestimmte Stelle des Bildschirms gesetzt werden.

Befehle zur Tonerzeugung

3 Tongeneratoren können mit BASIC-Befehlen angesprochen werden.

BEEP	erzeugt einen kurzen Ton
SOUND r,wert	schreibt Wert in Register r des Tongenerators
PLAY X$	spielt Musik entsprechend Zeichenkette X$.

Befehle für die Datenspeicherung

Ohne zusätzliches Disketten-Operationssystem ist sequentielle Datenspeicherung auf Kassetten mit folgenden Befehlen möglich:

OPEN"CAS:name" FOR mode AS #k	öffnet Kanal k zur Kassette
PRINT #k,ausdruck	Schreiben in Datei
PRINT#k USING,ausdruck	formatiertes Schreiben in Datei
INPUT #k,variablenliste	Lesen aus einer Datei
LINE INPUT #k,variable	Lesen bis Zeilenende
CLOSE k	Schließen des Kanals k.

Als Mode sind zugelassen **OUTPUT, INPUT, APPEND.**

BLOAD BSAVE	Laden und Speichern binärer Dateien.

Besonderheiten

Bildschirmeditor für Programmkorrekturen vorhanden. * Keine Rundungsfehler bei Rechnungen, da mit BCD-Arithmetik gerechnet wird. * Es besteht eine Reihe von weiteren Befehlen für die Bedienung von Steuerknüppeln und Drehreglern für Spiele. * Der eingebaute Zeitgeber und die Funktionstasten können im Programm abgefragt werden. * Programmierung von Sprites, Farbgrafik und Tonerzeugung in BASIC ohne POKE-Befehle möglich.

5.14 Das BASIC des ORIC-Atmos

Das ORIC-BASIC ähnelt in vieler Hinsicht dem Applesoft-BASIC, bietet aber noch einige zusätzliche Befehle. Insbesondere sind die Möglichkeiten für Grafik- und Farbdarstellung und für die Tonerzeugung weiter ausgebaut.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl, aber nur 2 Zeichen signifikant.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen. * Für »wahr« wird +1, für »falsch« 0 erzeugt. * Keine Einzelbitverarbeitung in BASIC. * Positive Zahlen werden ohne Leerstelle ausgegeben und gespeichert.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung. * Nicht vorhandene Befehle: INP, LINE INPUT, MOD, OUT, PRINT USING, SWAP, VARPTR, EOF, INSTR, STRING$, AUTO, RENUM. * Bei GET wartet das Programm unbeschränkt auf eine Eingabe.

Wichtige zusätzliche BASIC-Befehle

DEEK(adr)	liefert Inhalt zweier Adressen als Dezimalzahl
DOKE adr, a	speichert a in adr. und adr. +1
GRAB	gibt Reservierung für Hochauflösung frei
HEX$(a)	wandelt a in Hexadezimalzahl
HIMEMz	bestimmt höchsten Speicherplatz für BASIC

KEY$	wie GET, aber ohne Anhalten
POP	löscht letzte Stückeintragung
REPEAT...UNTIL	Schleifenanweisung
RELEASE	hebt GRAB auf
WAIT a	a/100 Sekunden warten.

Bildschirmaufteilung

Im Textmodus werden 27 Zeilen mit je 38 Zeichen gezeigt. Die beiden ersten Spalten in jeder Zeile dienen zum Speichern des Farbcodes. Bei hochauflösender Grafik sind 200×240 Punkte darstellbar, zuzüglich 3 Textzeilen unten.

Bildschirmcode

1. Zeichensatz (mit **LORES 0** aufrufbar)

0– 31	Steuercode
32–127	wie ASCII-Code, internationale Version.

2. Zeichensatz (mit **LORES 1** aufrufbar)

0– 31	Steuercode
32–127	Grafikzeichen.

Grafik- und Farbbefehle

In der Betriebsart LORES 1 können Grafiken mit den Zeichen des 2. Zeichensatzes mit PRINT-Befehlen erstellt werden. Hierfür gelten folgende Befehle:

PLOT x,y,p	setzt Zeichen p in x,y
PLOT x,y,X$	schreibt Zeichen des Strings, X$ ab x,y.

Wahl der Farbe und besonderer Darstellung (Blinken usw.) durch ESCAPE-Sequenzen, die unsichtbar jeweils als erste Positionen in jeder Zeile gespeichert werden.

In der Betriebsart HIRES (hochauflösende Grafik) gelten folgende Befehle:

CURSET x,y,p	setzt Cursor auf x,y und bestimmt Parameter für Vorder- und Hintergrundfarbe
CURMOVE x,y,p	bewegt Cursor relativ zur letzten Position
DRAW x,y	zeichnet Linie von Cursorposition nach x,y
FILL z,b,f	füllt z Zeilen mit b Byte in Farbe f
CIRCLE r,p	zeichnet Kreis mit Radius r
PATTERN a	bestimmt Darstellung von Linien
POINT (x,y)	prüft Grafikpunkt
PAPER h	schaltet Hintergrundfarbe ein
INK v	schaltet Vordergrundfarbe ein.

Befehle für die Tonerzeugung

Die Befehle **EXPLODE, PING, SHOOT** und **ZAP** erzeugen entsprechende Geräusche. Im übrigen stehen 3 Tongeneratoren für die Musikerzeugung zur Verfügung, die mit BASIC-Befehlen angesprochen werden können.

MUSIC	in den nachfolgenden Parametern kann der Kanal, die Oktave, die Note und die Lautstärke bestimmt werden
PLAY	mischt Töne aus zwei Kanälen und bestimmt Hüllkurve
SOUND	vereinfachter Tonbefehl für Geräusche.

Befehle für die Datenspeicherung

Numerische Datenfelder können auf einer Kassette mit **STOREx** gespeichert und mit **RECALx** wieder gelesen werden. Als x ist der Feldname einzusetzen.

Besonderheiten

Bildschirmeditor für Programmkorrekturen vorhanden. * Viele Routinen der Maschinensprache auch in BASIC zugänglich.

5.15 Das BASIC des Schneider CPC 464

Das auf der Grundlage des Microsoft-BASIC von der englischen Firma Amsoft entwickelte Locomotive-BASIC hat einen sehr gut ausgebauten Befehlssatz, der vor allem auch benutzerfreundliche Grafik- und Tonbefehle bietet und auch für Steuerungsaufgaben geeignet ist.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Variablennamen in Groß- und Kleinschrift bis zu 40 Zeichen zulässig und signifikant.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen. * RESTORE ist auch mit Zeilennummer zulässig. * Leerstellen oder Satzzeichen müssen vor und nach Befehlswörtern stehen.

Befehle des erweiterten BASIC

Keine Einschränkungen, aber nur Wandlung zwischen Integer- und Real-Variablen.

Statt LPRINT → **PRINT#8** (Druckerausgabe).
Statt LLIST → **LIST#8** (Listing auf Drucker).

Wichtige zusätzliche BASIC-Befehle

AFTER a,b	Unterprogrammanruf nach Zeitablauf a auf Uhr b
BIN$(a)	liefert Binärzahl von a
CHAIN n,z	lädt Programm mit Namen n und startet es ab Zeile z
DEG	schaltet von Bogenmaß auf Gradrechnung um
DI	verhindert Interrupt außer durch BREAK
ERASE a	Aufheben von Felddimensionierungen
EI	hebt DI-Anweisung auf
EVERY a	Unterprogrammanruf alle a Zeitintervalle
HEX$(a)	wandelt a in Hexadezimalzahl
HIMEM	enthält höchste von BASIC benutzte Adresse
INKEY(a)	prüft Tastatureingabe
JOY(a)	meldet Stellung des Steuerknüppels a
KEY a,"X"	belegt Taste a mit Zeichenkette X
LOG 10(a)	liefert Logarithmus zur Grundzahl 10
LOWERS(A$)	wandelt in A$ Groß- in Kleinschrift
MAX(liste)	liefert Maximum der Zahlen in der Liste
MEMORY adr	setzt Obergrenze für BASIC
MIN(liste)	liefert Minimum der Zahlen in der Liste
RAD	schaltet von Grad- auf Bogenmaßrechnung
ROUND(a,b)	liefert a, auf b Stellen gerundet
TIME	enthält Zeit
UPPERS(A$)	wandelt in A$ Klein- in Großbuchstaben
VPOS	enthält vertikale Cursorposition
WAITadr,b	wartet, bis an adr Bitmuster b ansteht
WHILE..WEND	Schleifenbefehle
WIDTH x	bestimmt Breite der Druckerausgabe
XOR	Operator für exclusiv oder.

Bildschirmaufteilung

Der **MODE**-Befehl bestimmt folgende Bildschirmeinteilungen:

0 25 Zeilen mit 20 Spalten, 16 Farben 200×160 Punkte
1 25 Zeilen mit 40 Spalten, 4 Farben 200×320 Punkte
2 25 Zeilen mit 80 Zeichen, 2 Farben 200×640 Punkte.

Mit **WINDOW#n,l,r,o,u** kann ein Bildschirmfenster definiert werden, das mit **#n** in verschiedenen Befehlen ansprechbar ist.

Bildschirmcode

0–127	ASCII-Code, internationale Version
128–255	Grafikzeichen.

Grafik- und Farbbefehle

In allen 3 Bildschirmmodi gelten folgende Befehle (Koordinatennullpunkt ist links oben):

PLOT x,y,v	setzt Punkt in x,y in Vordergrundfarbe v
DRAW x,y,v	zeichnet Linie nach u,v in Farbe v
DRAWR x,y,v	zeichnet x und y Schritte relativ
ORIGIN x,y	bestimmt Standpunkt des Grafik-Cursors
XPOS YPOS	enthält Position des Grafik-Cursors
MOVE x,y	bewegt Grafik-Cursor nach x,y
MOVER x,y	bewegt Grafik-Cursor relativ
BORDER f<,g>	bestimmt Randfarbe f, die mit g wechseln kann
PAPER f<,g>	bestimmt Hintergrundfarbe f
INK f<,g>	bestimmt Vordergrundfarbe f.

Mit **SYMBOL n, "X$"** können Sprites in der Zeichenkette definiert und mit **LOCATE x,y : PRINT CHR$(n)** auf den Bildschirm gebracht werden.

Befehle zur Tonerzeugung

3 Tongeneratoren können mit BASIC-Befehlen angesprochen werden.

SOUND	erzeugt einen Ton entsprechend den nachfolgenden Parametern für Kanal, Tonperiode, Lautstärke, Lautstärkenvariation, Tonvariation, Geräuschfolge
ENT	variiert nachfolgenden Ton in seiner Höhe
ENV	variiert nachfolgenden Ton in seiner Lautstärke.

Befehle für die Datenspeicherung

Durch BASIC wird die sequentielle Datenspeicherung auf dem eingebauten Recorder unterstützt. Für die Kassettenspeicherung gelten folgende Befehle:

OPENIN"dateiname"	öffnet Datei zum Lesen
OPENOUT"dateiname"	öffnet Datei zum Schreiben
PRINT#n,var.liste	schreibt Daten in Datei
INPUT#n,var.liste	liest Daten von Datei
CAT	gibt Inhaltsverzeichnis aus.

Besonderheiten

Zwei Arbeitsgeschwindigkeiten für den eingebauten Recorder möglich. * Weitere BASIC-Befehle auch für Steuerungsaufgaben in Echtzeit. * Zeileneditor und etwas gewöhnungsbedürftiger Copy-Editor vorhanden.

5.16 Das Sharp-BASIC der MZ-Modelle

Das BASIC des MZ-700 entspricht im wesentlichen den auf den MZ-80-Modellen laufenden Versionen. Es ist allerdings durch Farb- und Grafikbefehle zum Ansteuern des eingebauten Plotters erweitert.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl, aber nur die beiden ersten Zeichen signifikant.

Befehle der BASIC-Hochsprache

Nicht vorhanden : ELSE. * Vergleichsoperatoren nur bedingt zum Vergleich von Zeichenketten wirksam. * LOG(a) liefert den dekadischen Logarithmus.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung. * Nicht vorhandene Befehle : LINE INPUT, MOD, SWAP, VARPTR, FIX, INSTR, TRON, TROFF.

Statt LOCATE	→ **CURSOR** (Druckposition bestimmen).
Statt LPRINT	→ **PRINT/P** (Druckerausgabe).
Statt LLIST	→ **LIST/P** (Listing auf Drucker).
Statt POS	→ **CSRH** (Cursorposition).
Statt ERR	→ **ERN** (Fehlercode).
Statt FRE(x)	→ **SIZE** (freier Speicherraum).

Wichtige zusätzliche BASIC-Befehle

CSRV	liefert vertikale Cursorposition
DEF KEY(a)=	belegt Funktionstaste a (MZ-700)

LIMIT adr	bestimmt höchsten Speicherplatz in BASIC
LN(a)	liefert natürlichen Logarithmus von a
MON oder **BEY**	Anruf Monitor, Verlassen der BASIC-Ebene
SWAP	tauscht Programme von Kassette und Speicher
USR(adr)	bestimmt Anfangsadresse für Unterprogramm.

Bildschirmaufteilung

Im Textmodus 25×40 Zeilen.
Im Grafikmodus 50×80 Pixel.

Mit **CONSOLE** läßt sich beim MZ-700 ein Bildschirmfenster definieren.

Bildschirmcode

0– 26	Großbuchstaben
32– 47	Zahlen und Satzzeichen
129–155	Kleinbuchstaben
170–175	deutsche Umlaute.

Alle anderen Codes sind mit Grafikzeichen belegt.

Grafik- und Farbbefehle

Neben der PRINT- und POKE-Grafik im Textmodus können einzelne Punkte mit BASIC-Befehlen gesetzt und gelöscht werden. Koordinatennullpunkt links oben.

SET x,y,	setzt Punkt bei x,y
RESET x,y	löscht Punkt bei x,y.

Beim MZ-700 gibt es noch besondere Befehle zum Ansprechen des wahlweise einbaubaren Farbplotters:

MODE TN	Textbetrieb mit 40 Zeichen/Zeile
MODE TL	Textbetrieb mit 26 Zeichen/Zeile
MODE TS	Textbetrieb mit 80 Zeichen/Zeile
MODE GR	Grafikbetrieb mit 480 Punkten/Zeile
PCOLOR f	wählt Zeichenfarbe f
LINE<a,>x,y	zeichnet Linie in der Art a nach x,y
RLINE	wie vor, aber relative Werte von x,y
MOVE x,y	bewegt Stift nach x,y
RMOVE	wie vor, aber relative Werte für x,y
PHOME	bewegt Stift in Ausgangsposition
GPRINT(g,l),X$	druckt X$ in Größe g und Lage l
AXIS	zeichnet und skaliert Koordinatenkreuz
CIRCLE	zeichnet Kreis oder Kreisabschnitt.

Befehle zur Tonerzeugung

Tonerzeugung ist mit BASIC-Befehlen auf dem MZ-80 programmierbar, wobei nicht die physikalischen Daten, sondern die Notenbezeichnungen verwendet werden. Diese können für mehrere hintereinander zu spielende Töne in eine Zeichenkette geschrieben und mit **MUSIC X$** aufgerufen werden. Jeder Ton kann darin wie folgt spezifiziert werden:

Oktave, Halbtonspezifikation, Tonbezeichnung, Dauer.

Der Befehl **TEMPO** a bestimmt das Tempo der Tonfolgen.

Befehle für die Datenspeicherung

Zur Datenspeicherung auf einer Kassette muß ein spezieller Recorder verwendet werden. Es gelten dann folgende Befehle:

WOPEN"name"	öffnet Datei name zum Schreiben
PRINT/T a,b,X$	schreibt Daten auf Datei name
ROPEN"name"	öffnet Datei name zum Lesen
INPUT/T a,b,X$	liest Daten in Variable.

Bei Diskettenspeicherung ist auch noch die Dateinummer und das Laufwerk in den Öffnungsbefehl einzusetzen.

Besonderheiten

Bildschirmeditor für Programmkorrekturen vorhanden. * Cursor-Steuerkommandos können auch im Programm nach PRINT als Zeichenkette verwendet werden. * Der beim MZ-700 einbaubare Farbplotter kann statt eines Bildschirmgerätes zur Ausgabe verwendet werden.

5.17 Das Sinclair BASIC

Während das BASIC des ZX-81 recht knapp mit Befehlen ausgestattet ist, sind beim Spectrum die Grundbefehle im wesentlichen vorhanden. Beide Dialekte zeichnen sich aber durch gewisse Eigenarten aus, die sonst bei BASIC nicht üblich sind.

Variable und Variablennamen

Zahlenvariable mit 8 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Namen von Zahlenvariablen in unbegrenzter Länge signifikant. * Als Laufvariable in FOR...NEXT-Schleifen und für alle Felder nur ein Buchstabe als Name zulässig. * Zeichenketten müssen in Feldern gespeichert, und die Länge muß als erstes dimensioniert werden.

Grundbefehle der BASIC-Hochsprache

Nicht vorhanden beim ZX-81: READ, DATA, ELSE, ON GOTO, RESTORE, FRE. * Nach Funktionen wird das Argument ohne Klammer geschrieben. * Mehrfachzuweisungen mit LET sind erlaubt. * Neudimensionierung mit DIM ist erlaubt.

Statt ASC(x)	→ **CODE** x (Ausgabe Zeichencode)
Statt MID$, LEFT$, RIGHT$	→ Stringaufteilung mit **A$(b TO c)**
Statt RANDOMIZE	→ **RAND** (Neusetzen Zufallsgenerator).

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung und zur Fehlerbehandlung. * Nicht vorhandene Befehle bei beiden Modellen : LINE INPUT, MOD, PRINT USING, SWAP, VARPTR, EOF, FIX, INSTR, POS, STRING$, AUTO, DELETE, RENUM, TRON, TROFF. Darüber hinaus beim ZX-81 nicht vorhanden: DEF FN, VERIFY, OUT.

Wichtige zusätzliche BASIC-Befehle

ACS g	liefert Arcuscosinus von g
ASN g	liefert Arcussinus von g
BIN	Vorzeichen für Binärzahl (nur Spectrum)
BRIGHT	steuert Helligkeit der Ausgabe (nur Spectrum)
COPY	Hardcopy des Bildschirms auf dem Drucker
FAST	schneller Arbeitsrhythmus (nur ZX-81)
FLASH	Blinken der Ausgabe (nur Spectrum)
PAUSE a	Warten für a/50 Sekunden
SCROLL	eine Zeile weiterschieben (nur ZX-81).

Bildschirmaufteilung

ZX-81 Text	22 Zeilen mit je 22 Zeichen
ZX-81 Grafik	42×64 Punkte
Spectrum Text	22 Zeilen mit 32 Zeichen
Spectrum Grafik	176×256 Punkte.

Bildschirmcode

ZX-81		Spectrum	
1– 10	Grafikzeichen	0– 31	Steuercode
11– 27	Satz- und Sonderzeichen	32–127	wie ASCII-Code
28– 63	Ziffern und Buchstaben	128–143	Grafikzeichen
64– 66	Befehlswörter	144–164	frei für Benutzer

67–111	nicht benutzt	165–255	Befehlswörter
112–121	Befehlswörter		
122–191	wie 1–66, negativ		
192–255	Befehlswörter		

Grafik- und Farbbefehle

Nach dem Umschalten mit **GRAPHICS** können Grafikzeichen positiv oder negativ mit PRINT auf den Bildschirm gesetzt werden. Punktgrafik mit dem Koordinatennullpunkt links unten mit folgenden Befehlen:

PLOT x,y	setzt Punkt in x,y
UNPLOT x,y	löscht Punkt in x,y (nur ZX-81)
PLOT INVERSE	löscht Punkt (nur Spectrum)
DRAW x,y	zeichnet Linie mit x,y relativen Schritten (Spectrum)
CIRCLE x,y,r	zeichnet Kreis in x,y mir Radius r (nur Spectrum)

Nur der Spectrum erlaubt Farbdarstellung in 8 Farben.

Border f	bestimmt Farbe für den Rand
PAPER f	bestimmt Farbe für den Hintergrund
INK f	bestimmt Farbe für den Vordergrund.

Befehle für die Tonerzeugung

Tonerzeugung ist nur mit dem Spectrum möglich. Auf einem Tonkanal kann man mit dem Befehl **BEEP d, h** Töne in mehreren Oktaven erzeugen. Die Dauer d ist in Sekunden, die Tonhöhe in Halbtonschritten als positive oder negative Zahl einzusetzen.

Befehle für die Datenspeicherung beim Spectrum

Auf einer Kassette können mit **SAVE "name" DATA a(x)** auch Felder gespeichert und mit **LOAD "name" DATA a(x)** wieder in den Speicher eingelesen werden.

Für den Betrieb mit dem Microdrive gelten zum Speichern und Lesen von sequentiellen Dateien ähnliche Befehle, wie sonst beim Diskettenbetrieb üblich.

Besonderheiten

Die Prüfung auf Syntaxfehler findet sofort nach Betätigung der RETURN-Taste vor der Speicherung des Befehls statt. * Durch Benutzung der PEEK- und POKE-Befehle sind viele Program-

miertricks möglich. * In vielen Kleinigkeiten weicht das Laufverhalten trotz gleicher Befehlswörter von anderen BASIC-Dialekten ab.

5.18 Das BASIC der Spectra-Video-Computer

Das ebenfalls von Microsoft entwickelte BASIC für die Computer SV-318 und SV-328 ist sehr gut ausgebaut und benutzerfreundlich. Auch für farbige Grafikdarstellungen und klangvolle Musik sind keine POKE-Befehle notwendig.

Variable und Variablennamen

Zahlenvariable in einfacher und doppelter Genauigkeit. * Variablennamen in beliebiger Länge mit 2 signifikanten Zeichen.

Grundbefehle der BASIC-Hochsprache

Keine Einschränkungen. * Statt REM auch Apostroph zulässig. * RESTORE und RETURN auch mit Zeilennummer erlaubt.

Befehle des erweiterten BASIC

Keine Einschränkungen.

Wichtige zusätzliche BASIC-Befehle

BEEP	erzeugt kurzen Ton
BIN$(a)	liefert Binärwert von a (Vorzeichen $B)
CLICK ON/OFF	Tastaturklick ein-/ausschalten
CSRLINE	enthält Zeilenposition des Cursors
ERASE	löscht Dimensionierung von Feldern
HEX$(a)	liefert Hexadezimalwert von a (Vorzeichen $H)
KEY a,X$	setzt Taste a auf Wert XS
KEY LIST	Ausgabe der Bedeutung der Funktionstasten
MOTOR ON/OFF	schaltet Kassettenrecordermotor
OKT$(a)	liefert Oktalwert von a (Vorzeichen SO)
SCREEN m	bestimmt Bildschirmmodus
SWITCH	Umschalten auf zweite Speicherbank
TIME	enthält Zeit in 1/60 Sekunden
WAIT p,a	Warten, bis am Port p Bitmuster a ansteht.

Bildschirmaufteilung

SCREEN 0	Textmodus, je nach Einstellung mit **WIDTH** in 25 Zeilen, mit 39, 40 oder 80 Zeichen
SCREEN 1	hochauflösende Grafik mit 192×256 Punkten
SCREEN 2	niedrigauflösende Grafik mit 48–64 Punkten.

Bildschirmcode

0– 94	wie ASCII-Code 32–127
95–190	wie 32–127, aber negativ
191	Zeichen, auf dem der Cursor steht
192–247	Grafikzeichen, fest programmiert
248–255	selbst zu definierende Zeichen.

Grafik- und Farbbefehle

Für die hochauflösende Grafik (Koordinatennullpunkt links oben) gelten folgende Befehle, für die jeweils viele Parameter zugelassen sind:

LINE	zeichnet Linie absolut und relativ
DRAW X$	zeichnet Figur entsprechend X$
CIRCLE	zeichnet Kreise, Kreisabschnitte und anderes
PAINT	füllt Figur mit Farbe
COLOR	bestimmt Farbe für Vorder-, Hintergrund und Rand
GET	speichert Bildschirmbereich in ein Feld
PUT	gibt in einem Feld gespeicherte Grafik aus
LOCATE	schreibt Text in Grafik
SPRITE$	definiert Sprite
PUTSPRITE	setzt Sprite auf den Bildschirm.

Mit **VPEEK** und **VPOKE** sind auch direkte Eingriffe in das Videoram möglich.

Befehle zur Tonerzeugung

BEEP	erzeugt einen kurzen Ton
SOUND r,wert	schreibt Wert in Register r des Tongenerators
PLAY X$	spielt Musik, entsprechend Zeichenkette X$.

Befehle für die Dateiverwaltung

Unterstützt werden sequentielle Dateien und zusammen mit dem Disk-BASIC auch wahlfrei lesbare Dateien.

Schreibweise des Dateinamens: **"laufwerk:name.erg"**
Laufwerk=1 oder 2, Name 8 Zeichen, Ergänzung 3 Zeichen.

OPEN"name" FOR OUTPUT AS .n	Schreiben der sequentiellen Datei n
OPEN"name" FOR INPUT AS .n	Lesen der sequentiellen Datei n
PRINT #n,variablenliste	schreibt Daten in Datei n
INPUT #n,variablenliste	liest Daten aus Datei n in Variablenliste.

Für wahlfrei lesbare Dateien muß der Aufbau der Datensätze in einer Feldeinteilung festgelegt werden, wodurch ein Pufferspeicher reserviert wird. Es werden nur Zeichenketten gespeichert.

FIELD,n,25 AS Y$....	reserviert für Datenfeld Y$ 25 Byte
OPEN"name" AS #n	öffnet wahlfrei lesbare Datei
LSET Y$ = B$	schreibt B$ linksbündig in Feld Y$
RSET Y$ = B$	schreibt B$ rechtsbündig in Feld Y$
MKI$ MKS$ MKD$	wandeln Zahlen in Zeichenketten
CVI CVS CVD	wandeln Zeichenketten wieder in Zahlen
PUT# n,s	schreibt Daten vom Puffer n in Satz s
GET# n,s	holt Daten von Satz s in Puffer n
LOC(n)	liefert zuletzt bearbeitete Satznummer
LOF(n)	liefert Zahl der Sätze in der Datei n
BLOAD BSAVE	Laden und Speichern binärer Dateien.

Besonderheiten

Bildschirmeditor für Programmkorrektur. * Es gibt auch noch Befehle, die den Programmlauf nach einer Unterbrechung durch Steuerknüppel, Funktionstasten, Zeitgeber und anderes regeln.

5.19 Das BASIC des Thomson-TO 7

Das BASIC des TO 7 ist in erster Linie auf den Gebrauch im Heim, zum Spielen und für kreative Beschäftigung ausgelegt. Es ist benutzerfreundlich und ermöglicht gute Farbgrafik ohne POKE-Befehle.

Variable und Variablennamen

Zahlenvariable mit 9 Stellen. * Kein Variablentyp für doppeltgenaue Rechnung. * Variablennamen mit beliebiger Zeichenzahl, aber nur 16 Zeichen signifikant.

Befehle der BASIC-Hochsprache

Keine Einschränkungen. * RND erfordert kein Argument.

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung und zur Fehlerbehandlung. * Nicht vorhandene Befehle : DEFFN, INP, MOD, OUT, RANDOMIZE, PRINT USING, SWAP, CALL, VARPTR, MERGE, EOF, FIX, POS, STRING$, AUTO, RENUM. * Mit CLEAR muß Speicherraum für Zeichenketten reserviert werden.

Statt LLIST → **LIST"LPTR"** (Programmlisting auf Drucker)

Wichtige zusätzliche BASIC-Befehle

ATTRB	bewirkt vergrößerte Darstellung der Zeichen
BEEP	erzeugt kurzen Ton
CONSOLE 2	reserviert die ersten 2 Zeilen des Bildschirms.

Bildschirmaufteilung

Im Textmodus werden je 40 Zeichen in 25 Zeilen gezeigt oder wahlweise auch 200×320 Punkte gesetzt.

Bildschirmcode

0– 31	Steuercode
32–127	wie ASCII-Code, internationale Version
32–255	Grafikzeichen nach Umschaltung auf Grafik.

Die im Französischen üblichen Accents und die Doppelpunkte über den deutschen Umlauten können durch jeweils zwei zusätzliche Tastenanschläge erzeugt werden.

Grafik- und Farbbefehle

Für die PRINT-Grafik stehen 92 fest programmierte Zeichen mit je 6 Pixeln zur Verfügung. Außerdem können bis zu 128 Zeichen im 8-mal-8-Raster mit dem Befehl **DEFGRS(n)=** frei definiert werden. Hierfür ist auch Speicherreservierung mit **CLEAR anzahl** notwendig.

Vor dem Schreiben von Grafikzeichen mit der CHR$-Funktion muß mit **PRINT CHR$(14)** auf Grafikcode geschaltet werden. Die Rückschaltung auf den ASCII-Code geschieht mit **PRINT CHR$(15).**

Folgende Befehle dienen zum Zeichnen in hochauflösender Grafik:

SCREEN v,h,r	wählt Vorder-, Hintergrund und Rahmenfarbe
PSET x,y,	setzt Punkt bei x,y
LINE −(u,v)	zeichnet Linie nach u,v
LINE(x,y)−(u,v),f	zeichnet Linie von x,y nach u,v in Farbe f
BOX(x,y)−(u,v),f	zeichnet Rechtecke mit Ecken x,y und u,v
BOXF(x,y)−(u,v),f	zeichnet Fläche mit Ecken x,y und u,v.

Befehle für Benutzung des Lichtgriffels

Mit dem Lichtgriffel kann gearbeitet werden, wenn vorher die entsprechend markierte Stelle auf dem Bildschirm mit dem Griffel berührt wurde. Es gelten dann folgende Befehle:

INPUTPEN s,l	übergibt Koordinaten des Griffels an s und l
PSET (s,l)	setzt Punkt in s,l (Spalte, Linie)
LINE−(c,d)	zeichnet Linie, dem Griffel folgend
PEN n,(x,y)−(u,v)	definiert Feld n für Lichtgriffel
ON PEN n GOTO z	verzweigt Programm.

Befehle zur Tonerzeugung

Ein Tonkanal steht zur Tonerzeugung zur Verfügung, der mit den im Italienischen und Französischen üblichen Notenbezeichnungen DO – RE – MI – FA – SO – LA – SI – angesprochen wird. Die Notenbezeichnungen sind in eine Zeichenkette nach dem Befehlswort **PLAY** in der gewünschten Reihenfolge zu schreiben. Hier kann auch nur die Oktave (01 bis 05) und die Tondauer (L12−L96) eingesetzt werden.

Befehle für die Datenspeicherung

Zur Datenspeicherung auf einer Kassette muß immer ein Kanal geöffnet werden. Dann ist sequentielle Speicherung möglich.

OPEN,m,"name"	öffnet Datei mit dem genannten Namen auf dem Kanal n in der Betriebsart m auf dem Gerät g
PRINT #k,a,b,c	schreibt Daten auf Datei in Kanal k
INPUT #k,a,b,c	liest Daten von Kanal k in Variable.

Besonderheiten

Als Standardausrüstung eingebauter Lichtgriffel, der mit einfachen Befehlen programmiert werden kann. * Farbgrafik auch bei hoher Auflösung immer mit 8 Farben.

5.20 Das TI-BASIC von Texas Instruments

Das auf dem Homecomputer TI 99/4A laufende BASIC weicht in vielen Kleinigkeiten vom heute üblichen ab. Es gibt aber auch ein »Extended BASIC«, das zusätzliche Möglichkeiten für die Programmierung bietet. Da das TI-BASIC vor allem für Spielprogramme geschaffen wurde, bietet es viele Möglichkeiten für Grafik und Tonerzeugung.

Variable und Variablennamen

Zahlenrechnung im BCD-System, Darstellung mit 10 Stellen. * Exponentialdarstellung mit 6 Stellen. * Variablennamen mit maximal 15 Zeichen signifikant.

Befehle der BASIC-Hochsprache

Nicht vorhanden: SPC, LEFT$, RIGHT$. * Doppelpunkt nicht als Befehlsseparator zulässig. * Vor und hinter BASIC-Befehlswörtern müssen Leerzeichen beziehungsweise bestimmte Satz- und Sonderzeichen stehen.

Statt MID$ → **SEG$** (Aufteilung einer Zeichenkette).

Befehle des erweiterten BASIC

Keine Befehle zur Variablendefinition oder -wandlung. * Befehle für Fehlerbehandlung nur im EXT-BASIC. * Nicht vorhandene Befehle: INP, LINE INPUT, MOD, OUT, PRINT USING, SWAP, VARPTR, LPRINT, LLIST, INSTR, STRING$. * Keine Operatoren für logische Verknüpfungen.

Statt AUTO	→ **NUMBER** (Automatische Zeilennumerierung).
Statt CLS	→ **CALL CLEAR** (Bildschirm löschen).
Statt RENUM	→ **RESEQUENCE** (Neunumerierung).
Statt GET	→ **CALL KEY** (Tastaturabfrage).
Statt +	→ **&** (Stringverknüpfung).

Wichtige zusätzliche BASIC-Befehle

ACCEPT	positioniert Eingabe (nur EXT-BASIC)
BEY	Abschalten von BASIC
CALL JOYST	Abfrage der Steuerknüppel
DISPLAY	ähnlich PRINT, aber mit Positionierung
IMAGE	positioniert Ausgabe (nur EXT-BASIC)
NUMBER z	Aufruf des Zeileneditors
OPTION BASE	bestimmt ersten Index für Felder
PRINT :	macht Zeilenvorschub
UNBREAK	löscht alle mit BREAK markierten Stoppstellen.

Bildschirmaufteilung

Es werden je 32 (28) Zeichen in 24 Zeilen auf dem Bildschirm gezeigt.

Bildschirmcode

0– 31	Steuercode
32– 95	wie ASCII-Code, internationale Version
96–159	selbst definierte Zeichen.

Je 8 Codeziffern sind in einem Zeichensatz zusammengefaßt, um ihnen eine Farbinformation zuweisen zu können.

Grafik- und Farbbefehle

Alle Farb- und Grafikanweisungen erfolgen in Form von Unterprogrammanrufen mit **CALL programmname.** Die folgenden Programmnamen können verwendet werden:

COLOR(a,v,h)	bestimmt Vorder- und Hintergrundfarbe für die im Zeichensatz a stehenden Zeichen
SCREEN(h)	bestimmt Hintergrundfarbe n
CHAR(c,"string")	definiert im String Zeichen für Code n
HCHAR(z,s,c,w)	setzt das Zeichen mit Code c in Zeile z und Spalte s und wiederholt es w-mal in der Zeile
VCHAR(z,s,c,w)	wie HCHAR, aber mit Wiederholung in der Spalte
GCHAR(z,s,x)	speichert den Code des in Zeile z und Spalte s stehenden Zeichens in der Variablen x.

In dem String zur Definition von Zeichen im CHAR-Anruf müssen die zu setzenden Bitmuster als Hexadezimalzahlen geschrieben werden.

Befehle zur Tonerzeugung

Mit dem **SOUND**-Unterprogramm programmiert man die Dauer sowie die Tonhöhen und Lautstärken für 4 Töne. Die Dauer wird in 1/1000 Sekunden, die Tonhöhe als Frequenz in Hertz und die Lautstärke in 30 Stufen jeweils als Zahlenwert in den Befehl gesetzt. Durch Einsatz von negativen Zahlen als Frequenzwert kann auch Rauschen programmiert werden.

Befehle für die Datenspeicherung

Das TI-BASIC unterstützt sequentielle Dateien und mit Einschränkung auch wahlfreies Lesen im Kassettenbetrieb. Es werden hierfür zwei Recorder angeschlossen, einer für die zu lesenden und einer für die zu speichernden Daten.

OPEN #n:g,t,f,m	öffnet Datei mit der Nummer n auf dem Gerät g des Typs t im Format f in der Betriebsart m
PRINT #n: a,b,c	schreibt Daten auf Datei n aus Variablen
INPUT #n: a,b,c	liest Daten aus der Datei n in Variable.

Dateien mit wahlfreiem Zugriff werden durch das Wort **RELATIVE** als Dateityp gekennzeichnet. In den Schreib- und Lesebefehlen ist statt der Variablenliste **REC satznummer** einzusetzen.

Besonderheiten

Das TI-BASIC enthält noch einige besondere Befehle, mit denen der Computer im Direktbetrieb als »Formelrechner«, d. h. wie ein programmierbarer Taschenrechner, verwendet werden kann. * Zur Korrektur von Programmzeilen kann ein Zeileneditor angerufen werden.

6. Fehlende BASIC-Befehle ersetzen

Wie Kapitel 5 zeigt, sind einige BASIC-Befehle nicht in allen BASIC-Dialekten vorhanden. Wer ein für einen fremden Computer geschriebenes Programm auf seinem Gerät zum Laufen bringen will, muß darum wissen, wie er sich gegebenenfalls helfen kann, wenn sein Computer einige fremde Befehle nicht ausführt.

Leider gibt es aber auch BASIC-Befehle, die nicht zu ersetzen sind. Dies kann daran liegen, daß die Hardware der einzelnen Computer nicht übereinstimmt. Es kann aber auch an unterschiedlicher Betriebs-Software und einfach an mangelndem Wissen über den inneren Aufbau von Hard- und Software liegen.

Dieses Kapitel gibt Hinweise, wo ein Umschreiben von Befehlen leicht möglich ist, und wo man andere Wege bei einer Programmkonvertierung suchen muß.

6.1 Ersatz fehlender Programmanweisungen

IF...THEN...ELSE

Das Befehlswort ELSE in einem Entscheidungsbefehl als Verzweigungsanweisung wird von vielen BASIC-Dialekten nicht verstanden. Programme, die diesen Befehl enthalten, können aber umgeschrieben werden, wie das folgende Beispiel zeigt.

```
100 IF A>0 THEN PRINT"gut"ELSE PRINT"schlecht"

100 IF A>0 THEN PRINT"gut": GOTO 120
110 PRINT"schlecht"
120 .....
```

Es ist lediglich eine Zeile mit der Anweisung für falsches Vergleichsergebnis hinzuzufügen und die Anweisung für richtiges

Ergebnis mit einem Sprungbefehl abzuschließen, durch den die Zeile mit der Anweisung für »falsch« bei wahrem Vergleich übersprungen wird.

LINE INPUT

Bekanntlich können mit dem INPUT-Befehl bestimmte Sonderzeichen wie Komma, Doppelpunkt usw. nicht eingegeben werden. Bei einigen BASIC-Versionen gibt es darum noch den LINE INPUT-Befehl, mit dem eine Zeichenkette auch mit allen Sonderzeichen erstellt werden kann, bis RETURN die Eingabe beendet. Bei diesem Befehl bleibt auch die Wirkung der Löschtaste erhalten.

```
100 LINE INPUT A$
110 PRINT A$
100 E$=INKEY$: IF E$="" THEN 100
110 PRINT E$;: A$=A$+E$
120 IF ASC(E$)=8 THEN A$=LEFT$(A$,LEN(A$)-2)
130 IF ASC(E$)=13 THEN PRINT A$: GOTO 150
140 GOTO 100
150 ...
```

LINE INPUT läßt sich durch ein kleines Programm, das den GET- oder den INKEY$-Befehl verwendet, ersetzen. Normalerweise holt INKEY$ oder GET nur jeweils ein Zeichen. Wenn man einen String eingeben will, muß dieser zusammengesetzt werden. Dabei ergibt sich die Schwierigkeit, daß bei einer Falscheingabe mit der Löschtaste zwar ein Zeichen auf dem Bildschirm gelöscht wird, in der gespeicherten Zeichenkette aber stehenbleibt.

Wenn die Löschtaste betätigt wurde – hier ASCII-Code 8 – wird die Zeichenkette um zwei Zeichen verkürzt, denn außer dem zu löschenden Zeichen muß auch noch der Löschcode entfernt werden. Die RETURN-Taste mit dem Code 13 dient zum Abbruch der Schleife zum Zusammensetzen von A$.

PRINT USING

Der oft sehr nützliche Befehl für Rundung und formatierte Ausgabe kann auch durch andere Befehle ersetzt werden. Es gibt hierfür eine ganze Reihe von Programmvorschlägen. Einer davon wird im folgenden gezeigt.

```
100 PRINT TAB(20) USING "####.##":A

100 A$=STR$(INT(A*100+.5))
110 L=LEN(A$)
120 A$=LEFT$(A$,L-2)+"."+RIGHT$(A$,2)
130 PRINT TAB(20-L);A$
```

Entsprechend dem PRINT USING-Befehl soll die Variable A mit 4 Vorkomma- und 2 Nachkommastellen ausgegeben werden. In Zeile 100 wird sie mit 100 multipliziert, dann zur Rundung 0,5 addiert und daraus die Ganzzahl gebildet. Das Ergebnis wird mit Hilfe der STR$-Funktion in eine Zeichenkette gewandelt.

In Zeile 120 werden die beiden letzten Stellen abgetrennt; sie bilden die Nachkommastellen der eingegebenen Zahl, während der Rest den Ganzzahlwert darstellt. Nun braucht nur noch eine Zeichenkette zusammengesetzt zu werden, die aus diesen beiden Teilen und dem Zeichen für die Dezimaltrennung besteht. Ausgegeben werden muß dann die zusammengesetzte Zeichenkette an einer berechneten Tabulatorposition.

Wenn die formatierte Ausgabe mehrfach gebraucht wird, ist es zweckmäßig, hierfür ein Unterprogramm zu schreiben.

DEF FN

Viele BASIC-Dialekte bieten dem Benutzer die Möglichkeit, Rechenausdrücke als Funktionen im Programm zu definieren und diese dann durch Kurzbefehle aufzurufen. Wenn dieser Befehl bei einer BASIC-Version nicht vorhanden ist, sollte man bei einer beabsichtigten Programmkonvertierung zunächst untersuchen, wie oft die zu definierende Funktion vorkommt. Wenn dies nur ein- oder zweimal der Fall ist, wird es meist einfacher sein, in die betreffenden Programmzeilen statt des Funktionsaufrufs den Rechenausdruck selbst zu schreiben.

```
100 DEF FN K(D)=3.14159*D*D/4
110 INPUT A
120 PRINT FN K(A)

100 INPUT A
110 K = 3.14158*A*A/4
120 PRINT K
```

Wenn es sich aber um einen sehr komplexen Ausdruck handelt, oder wenn die Rechnung oft gebraucht wird, kann auch ein entsprechendes Unterprogramm geschrieben werden.

Der MOD-Operator

Einige BASIC-Versionen kennen einen besonderen Operator zur Bestimmung des Divisionsrests. Wird dieser gebraucht, kann man einfach einen Ersatzbefehl formulieren.

```
100 PRINT 2345 MOD 40

100 PRINT 2345-40*INT(2345/40)
```

Wie zu erkennen ist, wird der Ganzzahlwert des Quotienten mit dem Divisor multipliziert. Das Ergebnis, vom Dividenden abgezogen, ergibt den Rest. Ein falsches Ergebnis entsteht aber bei negativem Dividenden. Bei vielen BASIC-Versionen gibt es noch die FIX-Funktion. Wird diese statt INT eingesetzt, sind die Ergebnisse auch im negativen Bereich richtig. Wenn die FIX-Funktion nicht vorhanden ist, muß bei negativen Zahlen zum Ergebnis noch + 1 addiert werden.

WHILE . . . WEND und REPEAT . . . UNTIL

Einige BASIC-Dialekte bieten neben FOR...NEXT weitere Befehle zum Aufbau von Schleifen. Für die Bildung einer Schleife, bei der die Abbruchbedingung zu Beginn geprüft wird, aber keine Veränderung einer Laufvariablen stattfindet, werden die auch in anderen Programmiersprachen üblichen Befehle WHILE und WEND verwendet. Die gleiche Konstruktion kann leicht mit dem IF . . . THEN-Befehl aufgebaut werden.

```
100 F = 1
110 WHILE F < 1000
120 A = A+1 : F = F*A
130 PRINT A
140 WEND

100 F = 1
110 IF F>=1000 GOTO 150
120 A = A+1 : F = F*A
130 PRINT F
140 GOTO 110
150 ...
```

In der Schleife werden die Fakultäten gerechnet, und die Schleife wird abgebrochen, wenn ein Ergebnis über 1000 vorliegt. Bei Befehlsschreibung mit IF . . .THEN muß lediglich der Vergleichsoperator geändert werden.

Ähnliches gilt für die andere, gelegentlich verwendete Schleifenkonstruktion mit den Befehlen REPEAT . . .UNTIL. Hier muß die Abbruchsbedingung nach UNTIL formuliert werden; sie wird am Schluß des Schleifenlaufs geprüft.

```
100 F = 1
110 REPEAT
120 A = A+1 : F = F*A
130 PRINT F
140 UNTIL F>= 1000

100 F = 1
110 A = A+1 : F = F*A
120 PRINT F
130 IF F>=1000 GOTO 150
140 GOTO 110
150 ...
```

Der Tauschbefehl SWAP

Der bei neueren BASIC-Dialekten oft vorhandene Befehl SWAP, mit dem der Tausch zweier Variablen veranlaßt werden kann, ist für Sortiervorgänge gut zu verwenden. Er kann aber auch ersetzt werden, wenn man noch eine Hilfsvariable einführt.

```
100 SWAP A.B

100 C=A : A=B : B=C
```

LOCATE oder PRINT AT

Für eine bedienerfreundliche Bildschirmgestaltung ist es wichtig, den Cursor durch Programmbefehle an bestimmte Bildschirmpositionen setzen zu können, ohne daß dadurch andere Teile gelöscht werden. Bei einigen Dialekten können hierfür die Codes der Cursortasten in Zeichenketten gespeichert und dann mit PRINT-Befehlen im Programm zur Wirkung gebracht werden. Bei dieser Methode entstehen aber unschöne Ausdrucke im Listing, wenn die Codes überhaupt vom Drucker beachtet werden. Neuere Dialekte verwenden darum den LOCATE-Befehl mit nachfolgender Zeilen- und Spaltennummer zur Positionierung des Cursor.

Zwei einfache Methoden gibt es, den LOCATE-Befehl zu ersetzen.

```
100 LOCATE Z,S: PRINT"Humboldt"

100 CLEAR 1000: PRINT CHR$(28);
110 FOR K = 1 TO Z: PRINT CHR$(26);: NEXT
120 FOR K = 1 TO S: PRINT CHR$(25);: NEXT
130 PRINT "Humboldt"
```

In praktisch allen BASIC-Dialekten gibt es Steuercodes für die Bewegung des Cursor. Diese können in eine Schleife geschrieben werden, die so oft laufen muß, wie Schritte in vertikaler und horizontaler Richtung notwendig sind. Wichtig ist, daß die PRINT-Befehle mit einem Semikolon abgeschlossen werden, um den sonst üblichen Zeilenvorschub zu verhindern. Ob die beiden Steuercodes auch in einen Befehl geschrieben werden können, muß ausprobiert werden.

Wenn die STRING$-Funktion vorhanden ist, die jeweils eine Zeichenkette mit einer bestimmten Anzahl gleicher Zeichen erzeugt, kann man diese ebenso zum Ersatz von LOCATE verwenden.

```
100 CLEAR 1000: PRINT CHR$(28);
110 PRINT STRING$(Z,26);
120 PRINT STRING$(S,25);
130 PRINT "Humboldt"
```

6.2 Ersatz fehlender Funktionen

Die wirklich benötigten Funktionen sind praktisch bei allen BASIC-Dialekten vorhanden. Man wird darum nur selten Anlaß haben, etwas umzuschreiben, wenn man Programme konvertieren will. Trotzdem gibt es auch hier Ausnahmen.

Zeichenkettenveränderung

Es gibt einige Dialekte, bei denen die Funktionen LEFT$ und RIGHT$ fehlen. Sie lassen sich aber gut mit Hilfe der MID$-Funktion (die auch SEG$ heißen kann) und der LEN-Funktion ersetzen.

```
100 A$="Humboldt Taschenbuchverlag"
110 PRINT LEFT$(A$,8)
120 PRINT RIGHT$(A$,17)

RUN
Humboldt
Taschenbuchverlag
READY

100 A$="Humboldt Taschenbuchverlag"
110 PRINT MID$(A$,1,8)
120 PRINT MID$(A$,LEN(A$)-16)

RUN
Humboldt
Taschenbuchverlag
READY
>
```

Auch wenn (z. B. beim Atari und Sinclair) statt der MID$-Funktion eine andere Möglichkeit für die Aufteilung von Zeichenketten besteht, kann LEFT$ und RIGHT$ immer ersetzt werden. Es ist zu schreiben:

beim Atari	statt LEFT$(A$,z)	→	**A$=A$(1,z)**
	statt RIGHT$(A$,z)	→	**A$=A$(LEN(A$)−z-+1,LEN(A$))**
beim Spectrum	statt LEFT$(A$,z)	→	**A$=A$(1 TO z)**
	statt RIGHT$(A$,z)	→	**A$=A$(LEN A$−z+1 TO LEN A$.**

Die STRING$-Funktion

Mit Hilfe dieser Funktion kann eine Reihe gleicher Zeichen in eine Zeichenkette gesetzt werden. Wenn sie nicht vorhanden ist, kann man sie durch eine FOR...NEXT-Schleife ersetzen.

```
100 PRINT STRING$(40."*")

100 FOR K = 1 TO 40: PRINT"*";: NEXT: PRINT
```

Die INSTR-Funktion

Mit dieser Funktion wird untersucht, an welcher Stelle einer Zeichenkette eine andere Zeichenkette, die darin enthalten ist, beginnt. Die Funktion wird nicht sehr häufig benötigt. Wenn sie gebraucht wird, ist sie durch eine Schleife unter Verwendung von MID$ leicht zu ersetzen.

```
100 A$="Taschenbuchverlag"
110 B$="buch"
120 PRINT INSTR(A$,B$)

RUN
 8
READY


100 A$="Taschenbuchverlag"
110 B$="buch"
120 FOR K = 1 TO LEN(A$)
130 IF MID$(A$,K.LEN(B$))=B$ THEN PRINT K: END
140 NEXT

RUN
 8
READY
```

Die SPC-Funktion

In einigen BASIC-Dialekten fehlt die SPC-Funktion, mit der eine bestimmte Anzahl von Leerstellen in PRINT-Befehlen erzeugt werden kann. Sie ist gegebenenfalls durch die universellere STRING$-Funktion zu ersetzen.

```
100 PRINT SPC(10);"Hans"

100 PRINT STRING$(10,32);"Hans"
```

6.3 Wo kaum Befehlsersatz möglich ist

Beim Umschreiben von Programmen wird man aber auch auf Befehle stoßen, die nicht zu ersetzen sind. Einige davon kann man einfach weglassen, andere lassen sich nur mit erheblichem Aufwand umgehen.

Befehle, die fortfallen können

Entbehrlich sind ohne Folgen alle Befehle, die nur negative Bildschirmausgabe erzeugen (wie zum Beispiel INVERSE) und die wieder zurückschalten (wie NORMAL). Das gleiche gilt für als Negativzeichen dargestellte Steuerkommandos (z. B. negatives R für REVERSE beim Commodore).

Mit gewissen Einschränkungen gilt dies auch für reine Formatierungsbefehle wie PRINT AT, LOCATE usw., wenn diese nur im Programm stehen, um die Ausgabe gefälliger erscheinen zu lassen. Man kann das Programm zunächst ohne diese Befehle zum Laufen bringen und dann später mit den Befehlen des eigenen Computers eine bessere Bildschirmgestaltung programmieren.

Es gibt allerdings auch Formatierungsbefehle, die für den ordnungsgemäßen Lauf eines Programms unerläßlich sind. Meist handelt es sich um Befehle, die den Cursor nach oben oder nach links bewegen. Beim Aufbau zum Beispiel einer Eingabemaske kann man auf sie nicht verzichten.

Weggelassen werden kann der nur selten verwendete Befehl OPTION BASE.

Befehle, die ganz umgeschrieben werden müssen

Schwierig wird es, wenn in einem Programm Grafik-Befehle stehen. Manchmal können diese durch die Grafik-Befehle des eigenen Computers ersetzt werden. Aber oft ist es zweckmäßiger, zu ergründen, was eigentlich auf dem Bildschirm geschehen soll, und dann zu entscheiden, ob dieser Teil entfallen kann.

Unangenehm sind POKE- und PEEK-Befehle. Manchmal kann man sie weglassen, wenn damit nur Computerfunktionen umgeschaltet werden. Sie können aber auch eine wichtige Bedeutung für das Programm haben; und dann enden meist alle Bemühungen. Nur wenn die Bedeutung der Befehle im Programm erklärt ist, oder wenn der Inhalt der angesprochenen Adressen bekannt ist, können sie durch Befehle des eigenen Computers ersetzt werden.

Umschreiben der Befehle ist häufig notwendig, wenn ein Drucker im Programm angesprochen werden soll, denn nicht alle BASIC-Dialekte kennen den hierfür vielfach verwendeten LPRINT-Befehl und verlangen statt dessen die Öffnung eines Ausgabekanals zum Drucker.

Fast stets vollständig zu ändern sind Befehlsfolgen, mit denen ein externer Speicher angesprochen wird (beginnend immer mit OPEN). Auch wenn vielfach bei den einzelnen Dialekten die gleichen Befehlswörter verwendet werden, gibt es doch in der Syntax bemerkenswerte Unterschiede.

Arbeit erfordert die Umstellung, wenn im Programm der Befehl RESTORE mit nachfolgender Zeilennummer verwendet wird und auf dem eigenen Computer nur der einfache RESTORE-Befehl zur Verfügung steht. Es ist dann notwendig, den Ablauf beim Lesen der DATA-Zeilen genau zu prüfen und gegebenenfalls zusätzlich DATA-Zeilen in das eigene Programm zu schreiben.

Werden in einem Programm mit dem Befehl KEY Funktionstasten definiert, ist eine erhebliche Programmänderung notwendig, wenn der eigene Computer keine solche Tasten hat. Man kann dann Abfragen mit GET oder INKEY$ einbauen und andere sonst nicht verwendete Tasten als Funktionstasten programmieren.

Wo kaum etwas zu machen ist

Wenn ein Computer keinen Zeitzähler besitzt, ist es ohne zusätzliche Hardware nicht möglich, Zeitsignale zu verarbeiten, es sei denn, man verzichtet während der Zeitbestimmung auf andere Computerarbeit.

Kaum zu ersetzen sind die Befehle für die Fehlerbehandlung, wenn der eigene Computer nicht dafür eingerichtet ist.

Als fast unmöglich erweist sich eine Konvertierung, wenn im Programm Routinen der Maschinensprache des dort verwendeten Interpreters abgerufen werden. Nur in seltenen Fällen sind diese bekannt. Man kann nur versuchen, den Zweck der Routine zu erraten und dafür etwas Entsprechendes für den eigenen Computer schreiben.

Auch wenn in einem Programm die Funktion VARPTR verwendet wird, handelt es sich immer um einen trickreichen direkten Speicherzugriff, der nur durch Befehle in Maschinensprache ersetzt werden kann.

6.4 Hilfen für Zahlenwandlungen

In BASIC können bei Rechenaufgaben usw. Dezimalzahlen eingegeben werden, und der Computer antwortet auch in diesem Zahlensystem. Die Umrechnung in Binärzahlen, mit denen er allein rechnen kann, erfolgt selbsttätig. Für gewisse Programmieraufgaben benötigt man aber auch Hexadezimal- oder Binärzahlen. Einige BASIC-Dialekte bieten darum die Möglichkeit, Binär- oder Hexadezimalzahlen einzugeben oder Zahlen aus einem System in ein anderes zu wandeln.

Da diese Befehle aber nicht allgemein vorhanden sind, dürfte ein kleines Programm für die Zahlenwandlung von Interesse sein.

Zur Wandlung einer Dezimalzahl in ein anderes System eignet sich besonders die Divisionsmethode, die beispielsweise bei der Dez-Bin-Wandlung der Zahl 123 wie folgt arbeitet.

```
123/2 = 61 Rest  1
 61/2 = 30 Rest  1
 30/2 = 15 Rest  0
 15/2 =  7 Rest  1
  7/2 =  3 Rest  1
  3/2 =  1 Rest  1

ergibt Binärzahl        1  1  1  0  1  1
```

Es ist zu erkennen, daß die Dezimalzahl so lange durch 2 geteilt wird, bis keine Teilung mehr möglich ist. Bei jeder Teilung wird der Rest notiert, der – in umgekehrter Reihenfolge gelesen – die Binärzahl ergibt.

Ein BASIC-Programm, das diese Methode verwendet, kann wie folgt aussehen:

```
10 INPUT"Dezimalzahl";D             Eingabe Dez-Zahl
20 R = D-2*INT(D/2)                 Berechnung Divisionsrest
30 D = INT(D/2)                     neuer Dividend
40 B$ = STR$(R)+B$                  Zusammensetzung Bin-Zahl
50 IF D <> 0 THEN 20                Bedingung für Schleifenabbruch
60 PRINT"Binärzahl=";B$             Ausgabe Bin-Zahl
```

In etwas abgewandelter Form kann die gleiche Methode auch zur Dez-Hex-Wandlung verwendet werden. Hierfür ist es notwendig, am Anfang des Programms eine Zeichenkette aus allen Hexadezimalzahlen zu bilden, um den Wert der Ziffernsymbole auf einfache Weise zu bestimmen.

```
10 Z$="0123456789ABCDEF"            String mit allen Hex-Ziffern
20 INPUT"Dezimalzahl";D             Eingabe Dez-Zahl
30 R = D-16*INT(D/16)               Berechnung Divisionsrest
40 D = INT(D/16)                    neuer Dividend
50 C$ = MID$(Z$,R+1,1)              Bestimmen der Hex-Ziffer
60 H$ = C$ + H$                     Zusammensetzen Hex-Zahl
70 IF D <> 0 THEN 30                Abbruchbedingung
80 PRINT"Hex-Zahl = ";H$            Ausgabe Hex-Zahl
```

Für die Wandlung von Binär- oder Hexadezimalzahlen in Dezimalzahlen ist die Additionsmethode gut geeignet, die wie folgt arbeitet:

```
1 2 3 4 5 6 7 8          ← Stelle
7 6 5 4 3 2 1 0          ← Exponent
------------------------------------------
0 1 1 1 1 0 1 1          = Binärzahl
                         → 1*2^6 =  64
                         → 1*2^5 =  32
                         → 1*2^4 =  16
                         → 1*2^3 =   8
                         → 1*2^1 =   2
                         → 1*2^0 =   1
                           -----------
          Dezimalzahl    =         123
```

Es ist zu erkennen, daß einfach die Dezimalwerte jeder Stelle zu der zu wandelnden Zahl addiert werden.

Ein BASIC-Programm, das nach dieser Methode arbeitet, kann wie folgt aussehen:

```
10 INPUT"Binärzahl":B$                    Eingabe Bin-Zahl
20 FOR K = 1 TO LEN(B$)                   jede Stelle einzeln prüfen
30 IF MID$(B$,K,1)="0" THEN 50            wenn Stelle 0, keine Rechnung
40 D = D+2Ä(LEN(B$)-K)                    Rechnung und Summe Stellenwert
50 NEXT
60 PRINT"Dez-Zahl =";D                    Ausgabe Dez-Zahl
```

Da in der MID$-Funktion die Stellen von links beginnend gezählt werden, kann die Laufvariable K dort als Stellenzähler verwendet werden. Im Rechenausdruck muß aber der Komplementwert von K als Exponent eingesetzt werden.

Diese Methode läßt sich auch für die Hex-Dez-Wandlung verwenden:

```
10 Z$="0123456789ABCDEF"                  String mit allen Hex-Ziffern
20 INPUT"Hexadezimalzahl";H$              Eingabe Hexzahl
30 FOR J = 1 TO LEN(H$)                   alle Hex-Stellen bearbeiten
40 FOR K = 1 TO 16                        Hex-Ziffernsymbole durchsuchen
50 IF MID$(Z$,K,1) <>                     wenn Symbol im String gefunden,
   MID$(H$,J,1) THEN 70                   dann rechnen
60 D=D+(K-1)*16Ä(LEN(H$)-J)               Berechnung Stellenwert
70 NEXT K
80 NEXT J
90 PRINT"Dez-Zahl =":D                    Ausgabe Dez-Zahl
```

Man kann die beiden Wandlungsmethoden leicht in einem Programm zusammenfassen, das dann alle möglichen Umwandlungen durchführt. In unserem Programm, Abb. 14, sind die folgenden Variablen verwendet:

S	Anzahl der Ziffernsymbole im Zahlensystem
D	Dezimalzahl
A	Hilfsvariable
R	Divisionsrest
X	Feldindex für Bezeichnung der Zählsysteme
J,K	Laufvariable
Z$	String mit Ziffernsymbolen
H$	String für Hex- oder Bin-Zahl

X$	leere Eingabevariable
C$	Hilfsvariable
S$(x)	Feld für Bezeichnung der Zählsysteme.

Je nach Wunsch kann eine Dezimal-, eine Hexadezimal- oder eine Binärzahl eingegeben werden. Erstere wird in eine Hexadezimalzahl, die beiden anderen werden in Dezimalzahlen umgerechnet. Die zweite Wandlung erfolgt, wenn nach der Ausgabe von »Weiter?« eine Taste angeschlagen wurde.

Um das Programm kurz zu halten, ist keine Hex-Bin- und keine Bin-Hex-Wandlung programmiert. Beides erfolgt über die Dezimalzahlen. Da es hierbei Rundungsfehler geben kann, wird in den Zeilen 320 und 420 ein Korrekturwert hinzuaddiert.

```
10 REM Zahlenwandlungen ZAHLWAND
20 CLEAR 2000 : Z$="0123456789ABCDEF"
30 DATA"Dez.Zahl ","Hex.Zahl ","Bin.Zahl "
40 FOR K=1 TO 3: READ S$(K): NEXT
100 CLS: PRINT"Bitte wählen Sie"
110 PRINT"1 Dezimalzahl eingeben"
120 PRINT"2 Hexadezimalzahl eingeben"
130 PRINT"3 Binärzahl eingeben"
140 PRINT"4 Programm beenden
150 INPUT X
160 ON X GOTO 200,300,400,180
170 GOTO 100
180 END
200 PRINT S$(X)::INPUT D
210 S=16: A=D: H$="": X=2: GOSUB 600
220 S=2: A=D: H$="": GOSUB 600
230 GOTO 100
300 PRINT S$(X);:INPUT H$
310 D=0: S=16: X=1: GOSUB 700
320 S=2: A=D+.5 : H$="": X=3: GOSUB 600
330 GOTO 100
400 PRINT S$(X):: INPUT H$
410 D=0: S=2: X=1: GOSUB 700
420 S=16: A=D+.5: H$="": X=2: GOSUB 600
430 GOTO 100
600 R=A-S*INT(A/S)
610 A=INT(A/S)
620 C$=MID$(Z$,R+1,1)
630 H$ = C$+H$
640 IF A <> 0 THEN 600
650 PRINT S$(X);H$
660 INPUT"Weiter":X$: RETURN
700 FOR J = 1 TO LEN(H$)
710 FOR K = 1 TO S
720 IF MID$(Z$,K,1) <> MID$(H$,J,1) THEN 740
730 D=D + (K-1) * SÄ(LEN(H$)-J)
740 NEXT K
750 NEXT J
760 PRINT S$(X);D
770 INPUT"Weiter";X$: RETURN
```

Abb. 14: Programm zur Wandlung von Zahlen in andere Zählsysteme

7. Hilfreiche BASIC-Routinen verstehen

Die Sprachelemente von BASIC erlauben es, anstehende Programmieraufgaben auf sehr verschiedene Weise zu lösen. Man kann trickreich und effektiv, aber auch umständlich und langatmig programmieren. Wenn zu viele Programmtricks zur Anwendung kommen, wird ein Programm unverständlich, was aber auf keinen Fall heißt, daß jeder umständliche Programmaufbau auch gut verständlich ist.

In diesem Kapitel erklären wir einige BASIC-Routinen, die möglicherweise nicht gleich auf den ersten Blick verständlich sind. Sie können bei Bedarf leicht in eigene Programme eingebaut werden und dann andere, viel längere Befehlsfolgen ersetzen.

7.1 Aufforderungen zu Eingaben und deren Verarbeitung

Programm anhalten

Im Dialog mit dem Computer ist es oft erwünscht, dem Bedienenden Zeit zu geben, eine Anzeige genau zu lesen. Wenn die Darstellung mehr Zeilen umfaßt, als auf einem Bildschirm darstellbar sind, hilft ein kleines Programm, den Computer warten zu lassen.

```
100 FOR K = 1 TO 50
110 IF K/15=INT(K/15) THEN INPUT X$
120 PRINT A$(K)
130 NEXT
```

In Zeile 110 wird geprüft, ob der Schleifenzähler einen Wert erreicht hat, der eine Teilung durch 15 ohne Rest zuläßt. Denn dann ist das Ergebnis der Division eine Ganzzahl. Wenn dies der Fall ist, kommt der INPUT-Befehl zur Wirkung. Die dort stehende String-Variable hat keinerlei Bedeutung und dient, wie man

im Englischen sagt, als *Dummy*. Nach Betätigung einer beliebigen Taste läuft das Programm weiter und gibt wieder 15 Zeilen aus.

Falls der verwendete BASIC-Dialekt nach einem INPUT-Befehl das Programm abbricht, wenn nur die RETURN-Taste betätigt wurde, muß statt INPUT der GET-Befehl verwendet werden. Im allgemeinen ist aber INPUT einfacher.

Ja/Nein-Abfragen

Wenn der unterschiedliche Ablauf eines Programms durch eine Ja/Nein-Abfrage gesteuert werden soll, kann es zweckmäßig sein, einen Ablauf zu bevorzugen. In diesem Fall braucht nur eine Eingabe geprüft zu werden. Jede andere Eingabe bewirkt dann den alternativen Programmlauf.

```
100 INPUT"Drucker vorbereitet <J>":X$
110 IF X$="J" THEN 500
120 GOTO 100
```

Andere Überlegungen gelten, wenn sichergestellt werden soll, daß auch mehrere mögliche Eingaben vom Computer richtig erkannt werden. Das folgende Programm wertet die Eingaben j, J, ja und JA als Ja-Eingabe, ohne daß 4 Oder-Verknüpfungen notwendig sind. Denn die ASC-Funktion liefert immer nur den ASCII-Wert des ersten Zeichens eines Strings.

```
100 INPUT"Drucken":X$
110 IF ASC(X$)=74 OR ASC(X$)=106 THEN 200
120 IF ASC(X$)=78 OR ASC(X$)=110 THEN 300
140 GOTO 100
```

Menüauswahl mit Buchstaben

Wenn mehrere Wahlmöglichkeiten geboten werden, ist die Menü-Technik zweckmäßig. Am einfachsten wird das Programm, wenn der Benutzer aufgefordert wird, vor den Menü-Positionen stehende Kennziffern einzugeben. Diese können dann direkt in einem ON...GOTO-Befehl verwendet werden.

Die Bedienung wird allerdings etwas erleichtert, wenn statt der Zahlen kennzeichnende Buchstaben als Eingabe verwendet werden können. Für die Umsetzung der Buchstaben in die beim ON...GOTO-Befehl notwendigen Zahlen ist folgendes Programm nützlich:

```
9 REM Menüauswahl mit Buchstaben  MENUWAHL
100 PRINT"M e n ü a u s w a h l": PRINT
110 PRINT"    B = Bearbeiten
120 PRINT"    L = Löschen
130 PRINT"    A = Anfügen
140 PRINT"    N = Neueinrichten
150 PRINT"    E = Ende
160 INPUT X$
170 Y$="BLANE"
180 FOR K = 1 TO 5
190 IF X$=MID$(Y$,K,1) THEN 300
200 NEXT: PRINT"Unzulässige Eingabe": GOTO 100
300 ON K GOTO 1000,2000,3000,4000,5000
1000 'Unterprogramm für Bearbeiten
2000 'Unterprogramm für Löschen
3000 ' usw
```

In der Schleife wird geprüft, ob der eingegebene Buchstabe mit einem Buchstaben des aus den erlaubten Eingabebuchstaben gebildeten Strings Y$ übereinstimmt, und dann die Schleife abgebrochen. Die Laufvariable wird dann im ON...GOTO-Befehl verwendet.

Noch einfacher ist diese Programmaufgabe zu lösen, wenn die INSTR-Funktion zur Verfügung steht.

```
180 '
190 ON INSTR(Y$.X$) GOTO 1000.2000.3000...
200 PRINT"Unzulässige Eingabe": GOTO 100
```

Diese Funktion liefert den Platz, an dem der String X$ in Y$ beginnt.

Wandlung von Klein- in Großbuchstaben

Um bestimmte Daten nach dem Alphabet zu sortieren, müssen diese einheitlich entweder mit großen oder mit kleinen Buchstaben geschrieben sein. Bei vielen Computern gibt es darum auch eine Möglichkeit, die Tastatur so zu schalten, daß nur Großbuchstaben erzeugt werden. Wenn dies nicht vorgesehen ist, kann das folgende Hilfsprogramm zur Wandlung einheitlich in Kleinschrift nützlich sein:

```
10 REM Klein-Großwandlung KLEGROWA
90 INPUT"Gemischter String";G$
100 FOR K = 1 TO LEN(G$)
110 H$=MID$(G$,K,1)
120 IF ASC(H$) < 97 THEN C$=H$: GOTO 150
130 X = ASC(H$) -32
140 C$ = CHR$(X)
150 A$=A$+C$
160 NEXT
170 PRINT A$
```

In der Schleife wird jeweils ein Zeichen von dem gemischt eingegebenen String abgetrennt und untersucht, ob der dazugehörige ASCII-Code kleiner als 97 ist (denn dann muß es sich um einen Großbuchstaben handeln). Wurde ein größerer ASCII-Wert festgestellt, muß dieser um 32 erniedrigt werden, da dies den Unterschied zwischen Groß- und Kleinbuchstaben im ASCII-Code ausmacht. Es wird dann eine neue Zeichenkette aus den Einzelzeichen gebildet, wobei der aktuell berechnete ASCII-Code vorher noch in ein Zeichen gewandelt werden muß.

7.2 Bildschirmausgaben positionieren

Überschriften zentrieren

Oft ist es erwünscht, eine Überschrift oder ähnliches genau in die Mitte einer Bildschirmzeile zu setzen. Hierfür empfiehlt sich die Verwendung der TAB-Funktion mit einem Rechenausdruck als Argument.

```
50 A$="Programmwahl"
100 PRINT TAB(32-LEN(A$)/2):A$
```

Der auszugebende Text muß in einer Zeichenkettenvariablen vorliegen. Als Tabulatorwert wird die halbe Länge der Kette von der mittleren Bildschirmposition (hier also 32 bei 64 Zeichen/Zeile) abgezogen.

Rechtsbündige Ausgabe

Normalerweise werden Zahlen rechtsbündig und Texte linksbündig auf den Bildschirm gesetzt. Wenn die rechtsbündige Ausgabe von Zeichenketten gewünscht wird, verwendet man oft die folgende kleine Routine, die hier in einer Schleife steht:

```
50 L$="                    "
100 FOR K = 1 TO 5
110 INPUT A$
120 B$=RIGHT$(L$+A$,15)
130 PRINT B$
140 NEXT
```

Benötigt wird eine Zeichenkette, die nur eine Anzahl von Leerstellen enthält. Sie kann entweder mit der STRING$-Funktion oder, wie im Beispiel in Zeile 50, als L$ gebildet werden. Der Ausgabestring B$ wird dann aus L$ und dem Eingabestring A$ zusammengesetzt, hiervon mit RIGHT$ die benötigte Länge abgetrennt.

Formatierte Zahlenausgabe

Praktisch alle BASIC-Dialekte besitzen die Fähigkeit, auch Rechenausdrücke als Argument in der TAB-Funktion zu akzeptieren. Dies kann für formatierte Zahlenausgabe, bei der alle Dezimalpunkte untereinander stehen, verwendet werden.

```
10 REM Formatierte Ausgabe FORMAUSZ
50 INPUT"Punkt in welche Spalte";X
100 INPUT"Beliebige Zahl";Z
110 L=LEN(STR$(INT(Z)))
120 IF FIX(Z)=0 THEN L=L-1
130 PRINT TAB(X-L);Z
140 GOTO 100
```

Hier werden zunächst die Dezimalstellen der auszugebenden Zahl mit der INT-Funktion weggestrichen. Dann wird die Ganzzahl mit STR$ in eine Zeichenkette gewandelt. Die Länge dieser Kette, mit LEN bestimmt, ergibt, von der Druckposition des Dezimalpunktes abgezogen, die Position, an der die Ausgabe der Ziffer zu beginnen hat. Hierdurch kommen alle Dezimalpunkte untereinander zur Ausgabe. Diese Routine kann noch durch eine Rundung ergänzt werden. Mit der INT-Funktion ist sie allerdings nur für positive Zahlen, mit FIX statt INT für alle Zahlen brauchbar.

Zahlen in Kolonnen schreiben

Mit dem Komma im PRINT-Befehl ist zwar bei allen BASIC-Dialekten eine Kolonnenbildung möglich, die Anzahl der Kolonnen ist aber immer beschränkt. Wenn man eine größere Anzahl braucht, um z. B. viele kleine Zahlen auf einem Bildschirm darzustellen, kann folgendes Programm von Nutzen sein:

```
10 REM Zahlen in Kolonnen KOLONNEN
50 Z = 2
100 FOR K = 1 TO 6
110 Z=Z*3
120 A$="     "+STR$(Z)
130 A$=RIGHT$(A$,5)
140 PRINT A$;
150 NEXT
```

Die Schleife begrenzt die Anzahl der in einer Zeile darzustellenden Zahlen. Die Befehle in den Zeilen 50 und 110 dienen nur dazu, die Demonstrationszahlen zu erzeugen. In 120 wird ein String aus fünf Leerstellen und der in eine Zeichenkette gewandelten Ausgabezahl gebildet, der in 130 auf die Länge von 5 Zeichen beschnitten wird. Da der nachfolgende PRINT-Befehl, mit dem die Zeichenkette ausgegeben wird, am Schluß ein Semikolon hat, werden die Zahlen im Abstand von 5 Spalten auf den Bildschirm geschrieben.

Zeichnen eines Rahmens

Um eine Menü-Darstellung oder eine Maske auf dem Bildschirm besser zu präsentieren, wird diese oft in einen Rahmen gesetzt. Man kann dies auf verschiedene Weise programmieren. Drei Methoden werden hier erklärt.

```
10 REM Rahmen zeichnen RAHMEN
100 CLS: FOR K=1 TO 64: PRINT"*";: NEXT
110 FOR K=1 TO 14 : PRINT"*"TAB(63)"*";: NEXT
120 FOR K=1 TO 63: PRINT"*";: NEXT
130 A$="Aufgabenübersicht"
140 PRINT§ 128+32-LEN(A$)/2,A$:
150 GOTO 150
200 FOR K=15360 TO 15423
210 POKE K,143 : NEXT
220 FOR K=15360 TO 16320 STEP 64
230 POKE K,191 : NEXT
240 FOR K=15423 TO 16383 STEP 64
250 POKE K,191 : NEXT
260 FOR K=16320 TO 16383
270 POKE K,143 : NEXT
290 GOTO 290
300 CLEAR(1000): R$=STRING$(62.143)
310 PRINT§ 1,R$
320 FOR K=15360 TO 16320 STEP 64
330 POKE K.191 : NEXT
340 FOR K=15423 TO 16383 STEP 64
350 POKE K,191 : NEXT
360 PRINT§ 961,R$;
370 GOTO 370
```

Im ersten Teil des Beispiels verwenden wir den normalen PRINT-Befehl. Es sind drei FOR...NEXT-Schleifen geschrieben. Die erste und die dritte Schleife zeichnen die oberen und unteren Rahmenkanten, die zweite zieht beide Senkrechte, wobei die TAB-Funktion für die notwendigen Leerstellen sorgt. Gezeichnet wird der Rahmen mit dem Sternsymbol. Wichtig sind die Semikolons jeweils am Ende der PRINT-Befehle.

In einen Rahmen kann man nicht mit dem einfachen PRINT-Befehl schreiben, sondern muß den Cursor vorher in die gewünschte Position bringen. Hier wird der PRINT@-Befehl verwendet, um A$ mittig zu schreiben.

Wem der Aufbau eines Rahmens, wie eben beschrieben, zu lange dauert, der kann auch mit dem POKE-Befehl Grafikzeichen direkt in den Bildschirmspeicher setzen. Es müssen dann vier Schleifen, für jede Rahmenseite eine, geschrieben werden.

Noch schneller wird der Rahmen gezeichnet, wenn man nur die Senkrechten mit POKE auf den Bildschirm bringt und für die Waa-

gerechten mit der entsprechenden Zahl von Zeichen eine Zeichenkette bildet, die als obere und untere Begrenzung sehr schnell mit einem PRINT-Befehl ausgegeben werden kann.

7.3 Möglichkeiten bei Vergleichen

Formulierung von Vergleichsbedingungen

Nicht ganz leicht verständlich sind manchmal die in IF...THEN-Befehlen stehenden Vergleichsbedingungen. Es gibt hier immer mehrere Möglichkeiten der Formulierung, die überdies computerspezifisch voneinander abweichen können. Einige Beispiele werden im folgenden erklärt. In der gleichen Zeile stehen jeweils zwei Formulierungen mit gleicher Wirkung.

Bei allen BASIC-Dialekten wird 0 als Kennzahl erzeugt, wenn ein Vergleich als »falsch« erkannt wurde, während sowohl +1 als auch −1 für »wahr« üblich sind (siehe auch Test 25−27 in Abschnitt 4.7). Es gibt nun Funktionen, die nur zwei Zustände mit der Bereitstellung von 0 oder −1 melden. So ist es z. B. bei vielen BASIC-Dialekten möglich, nur die Funktion ohne voll formulierte Vergleichsbedingung in den IF...THEN-Befehl zu schreiben.

```
IF EOF(1) THEN..                IF EOF(1)= -1 THEN..
```

Verschachtelte Vergleiche, die in einigen Dialekten zulässig sind, können durch Verbindung mit dem AND-Operator ersetzt werden.

```
IF A=2 IF B>0 THEN..            IF A=2 AND B>0 THEN..
```

Wenn auch mit den Vergleichskennzahlen gerechnet werden kann, findet man manchmal statt einer AND- oder OR-Verknüpfung die Bedingung wie links gezeigt formuliert.

```
IF (A=2)+(B>0)= -2 THEN.. IF A=2 AND B>0 THEN..

IF (A=2)+(B>0)= -1 THEN.. IF A=2 OR B<0 THEN..
```

Um festzustellen, ob eine Zahl positiv oder negativ ist, kann man die SGN-Funktion verwenden. Das gleiche Ergebnis ist auch durch einen Vergleich mit 0 erzielbar.

```
IF SGN(A) = 1 THEN..       IF A>0 THEN..
```

Wenn bei als falsch erkanntem Vergleich 0 erzeugt wird, kann die AND-Verknüpfung durch das Multiplikationszeichen ersetzt werden, sofern die beiden Bedingungen davor und dahinter in Klammern stehen.

```
IF (A=2)*(B>0) THEN..     IF A=2 AND B>0 THEN..
```

Soll geprüft werden, ob mehrere Variable den Wert 0 haben, kann dies mit dem AND-Operator, aber auch durch eine additive Verknüpfung erfolgen. Ähnliches gilt, wenn man ermitteln will, ob eine von mehreren Variablen den Wert 0 hat. In diesem Fall kann statt mit dem OR-Operator mit Multiplikationen verknüpft werden.

```
IF A + B + C = 0 THEN..   IF A=0 AND B=0 AND C=0 THEN..

IF A * B * C = 0 THEN..   IF A=0 OR B=0 OR C=0 THEN..
```

Die Verwendung des NOT-Operators könnte etwas Verwirrung stiften, da jeder Vergleich damit umkehrbar ist. Man muß immer berücksichtigen, daß durch NOT das Bitmuster invertiert, d. h. daß aus Nullen Einsen und umgekehrt gemacht werden.

```
IF NOT A > NOT B THEN..   IF A < B THEN..
```

Vergleich mit Joker

Beim Suchen müssen bekanntlich das gesuchte und das eingegebene Wort genau übereinstimmen, wenn der Computer es mit einem Vergleichsbefehl finden soll. Es kommt aber immer wieder vor, daß man nicht genau weiß, wie ein gespeichertes Wort geschrieben wurde. Das folgende kleine Programm erlaubt darum die Einsetzung des Dollarzeichens als Joker in das Suchwort, an den Stellen, an denen die genaue Schreibweise nicht bekannt ist.

```
10 REM Vergleich mit Joker JOKER
50 INPUT"N$";N$
60 INPUT"NA$ Mit Joker";NA$
100 FOR K = 1 TO LEN(NA$)
110 IF MID$(NA$.K.1) <> "$" THEN 130
120 N$=LEFT$(N$,K-1)+"$"+MID$(N$,K+1)
130 NEXT
140 N$=LEFT$(N$,LEN(NA$))
150 PRINT "NA$=";NA$."N$=";N$
```

Im Programm wird angenommen, daß N$ gespeichert wurde und NA$ eingegeben wird. In der Schleife wird jedes Zeichen von NA$ untersucht, ob es den Charakter eines Jokers hat. Ist dies der Fall, wird in die gleiche Stelle von N$ auch ein Joker gesetzt. Vor dem Vergleich, der hier nur in Zeile 150 durch die Ausgabe beider Zeichenketten simuliert ist, wird der gespeicherte String noch auf die Länge des Eingabestrings beschnitten.

7.4 Den Speicherraum besser ausnutzen

Im allgemeinen werden zur Speicherung einer Ganzzahl 5 bis 7 Byte und zur Speicherung einer Zeichenkette 6 bis 7 Byte zuzüglich einem Byte für jedes Zeichen benötigt (siehe auch die Abb. 8 und 9 im Abschnitt 4.2, Seite 60 und 61). Sollen nur kleine Zahlen gespeichert werden, bedeutet dies eine gewisse Vergeudung von Speicherraum, die allerdings nur dann ins Gewicht fällt, wenn es viele Zahlen zu speichern gilt.

Wir kennen darum eine ganze Anzahl von Methoden, benötigte Daten dichter gepackt zu speichern. Einige davon werden im folgenden erklärt.

Zahlen als Bitmuster speichern

In einem Byte lassen sich 256 verschiedene Informationen unterbringen. Wenn ausschließlich Zahlen bis zu dieser Grenze gespeichert werden sollen, könnte man sie als Binärziffern ablegen, was aber nur in der Maschinensprache aussichtsreich ist. Es gibt jedoch die Möglichkeit, derartige Zahlen jeweils in einem Zeichen einer Zeichenkette zu speichern, wenn man den ASCII-Code dafür verwendet (der auch die Werte von 0 bis 255 annehmen kann).

```
10 REM Speicherplatz sparen    SPEISPAR
50 INPUT"Wieviele Zahlen";M
100 FOR K = 1 TO M
110 INPUT"Zahl 0...254";Z
120 A$=CHR$(Z)
130 B$=B$+A$
140 NEXT
200 FOR K = 1 TO M
210 C$=MID$(B$,K,1)
220 PRINT ASC(C$);
230 NEXT
```

In dem Programm läuft die Schleife so oft, wie einzelne Zahlen von 0–254 gespeichert werden sollen (der eingesetzte Computer akzeptiert den Codewert 255 nicht mehr). In Zeile 120 wird die CHR$-Funktion verwendet, um aus der eingegebenen Zahl das dazugehörige ASCII-Zeichen zu erzeugen, das der String-Variablen A$ zugewiesen wird. In Zeile 130 wird dann der Gesamtstring B$ aus den einzelnen Zeichen zusammengesetzt. Zur Kontrolle, daß das Programm richtig arbeitet, wird in 210 B$ wieder aufgeteilt und anschließend der ASCII-Code jeder Zahl wieder ausgegeben.

Drei Zahlen als eine Ganzzahl speichern

Für die Speicherung einer Ganzzahl benötigen die meisten BASIC-Dialekte 16 Byte, die der Computer für den Zahlenbereich von −32 768 bis +32 767 verwendet. Wenn statt dessen mehrere kleinere Zahlen zusammen gespeichert werden sollen, kann man diese auch in einer Integer-Variablen unterbringen.

```
10 REM 3 Zahlen in einem Integer  DREIZASP
20 DEFINT A,S,T,W
100 INPUT"Woche <1-52>";W
110 IF W > 52 THEN 100
120 INPUT"Tag <1-7>";T
130 IF T > 7 THEN 120
140 INPUT"Stunden <1-12>";S
150 IF S > 12 THEN 140
200 A = W*2^7 + T*2^4 + S
201 PRINT A
300 WO = A/128: TA = (A-WO*128)/16
310 ST = (A-WO*128-TA*16)
311 PRINT  WO,TA,ST
```

Das Beispiel geht von der Annahme aus, daß die Wochenzahl (1–52), der Wochentag (1–7) und eine Kennziffer für die Stunde (1–12) zu speichern sind. Für die Woche werden 7 Bit, für den Tag 4 Bit und für die Stunde 4 Bit verwendet. Das sind insgesamt 15 Bit, so daß das höchstwertige Bit, das meist als Vorzeichenbit in der im Computer verwendeten 2er-Komplementmathematik dient, nicht gebraucht wird.

Alle im Programm verwendeten Variablen werden zu Beginn als Integer-Variable definiert. Nach den Eingabebefehlen und der Eingabeprüfung wird in Zeile 200 die zu speichernde Ganzzahl A aus den drei Teilzahlen zusammengesetzt. Hierzu muß die Woche mit 2^7 multipliziert werden, weil die Wochenzahl ab dem Bit Nr. 7 gespeichert wird. Ebenso ist die Tageszahl mit 2^4 malzunehmen, da hier die Speicherung mit dem Bit Nr. 4 beginnt. Für die Speiche-

rung der Stundenzahl werden die Bits Nr. 0 bis 3 benutzt, für die kein Multiplikationsfaktor notwendig ist.

Die letzten Befehle zeigen die Rückwandlung vom Integer in die drei einzelnen Zahlen. Die Wochenzahl entsteht bei Division durch 128, den Wert von 2^7. Verwendet wird nur der ganzzahlige Anteil, da durch die Definierung auch WO zur Integer-Variablen erklärt wurde. Um den Tag zu ermitteln, muß der zur Darstellung der Wochenzahl verwendete Teil der Ganzzahl A von dieser abgezogen und der Rest durch 2^4, also durch 16, geteilt werden. Die Stundenzahl ergibt sich dann auf gleiche Weise durch Subtraktion der schon verwendeten Teile der Ganzzahl.

Gepackte String-Speicherung

Bei einigen Typen wahlfrei lesbarer Dateien muß man die Länge jedes Datenfeldes vorher festlegen. Wenn man nun stets von der möglichen Maximallänge jeder in ein Feld zu setzenden Eintragung ausgeht, führt dies zu Platzverschwendung: Es kommt wohl nie vor, daß bei allen Eintragungen die Maximallänge benötigt wird. Faßt man dagegen mehrere Einzeleintragungen in einem Datenfeld zusammen, kann dieses kürzer definiert werden, da sich lange und kurze Eintragungen ausgleichen.

```
10 REM Gepackte Speicherung PACKSPEI
70 N$="Maier"
80 V$="Hans-Helmut"
90 O$="Niederkutzenhausen-Großdorf"
100 X$=N$+","+V$+","+O$
200 Z=1: GOSUB 1000
210 N$=LEFT$(X$,K-1): PRINT N$
220 Z=K+1: GOSUB 1000
230 V$=MID$(X$,Z,K-Z): PRINT V$
240 Z=K+1: GOSUB 1000
250 O$=MID$(X$,Z): PRINT O$
990 END
1000 FOR K = Z TO LEN(X$)
1010 IF MID$(X$,K,1)="," THEN RETURN
1020 NEXT: RETURN
```

Im Beispiel wird der Name, der Vorname und der Ort zusammengefaßt, indem in Zeile 100 der Gesamtstring X$ aus den Teilstrings unter Zwischenschaltung von Kommas gebildet wird. Zur Aufteilung von X$ in die Teilstrings wird ein Unterprogramm verwendet. Dort steht eine FOR...NEXT-Schleife, in der der Anfangswert der Laufvariablen Z jeweils aus dem Hauptprogramm entnommen wird. In der Schleife werden die einzelnen Zeichen von X$ untersucht, und wenn dort ein Komma gefunden wird, erfolgt der Rücksprung zum Hauptprogramm. Dort geschieht dann mit Hilfe von LEFT$ bzw. von MID$ die Ausgabe der einzelnen Teilstrings.

7.5 Direkter Speicherzugriff

Dezimalzahl in 2-Byte-Binärzahl wandeln

Da einerseits in BASIC alle Zahlenein- und -ausgaben in dezimaler Form erfolgen, andererseits aber alle Zahlen im Binärformat abgespeichert werden, sind immer Umrechnungen notwendig, wenn die Binärzahl mehr als ein Byte benötigt. Bei den am meisten verwendeten Mikroprozessormodellen der Z 80-, 80xx- und 65xx-Serien werden 2-Byte-lange Binärzahlen grundsätzlich in der Weise gespeichert, daß das niederstwertige Byte in der niedrigsten Adresse und das höchstwertige Byte in der nächsthöheren Adresse gespeichert werden (Abb. 15).

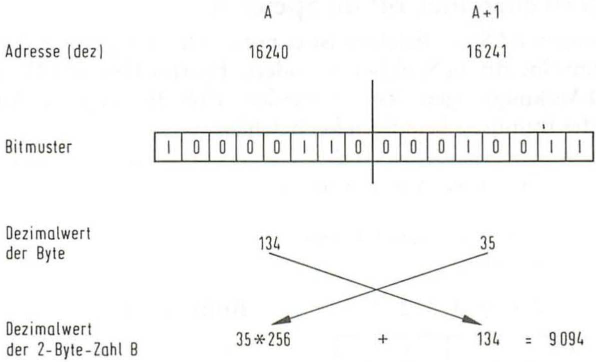


Abb. 15: Speicherung von Ganzzahlen in 2 Byte
Bei den in Personal-Computern vielfach verwendeten Microprozessoren der Typen 8085, Z 80 und 65xx wird das höherwertige Byte der Binärzahl auch in der höheren Adresse gespeichert. Bei der Umrechnung in eine Dezimalzahl müssen darum die Bytes gekreuzt werden

Wenn also z. B. eine Zeilennummer oder eine Adresse mit PEEK aus dem Speicher gelesen beziehungsweise mit POKE in den Speicher geschrieben werden sollen, müssen die Befehle wie folgt formuliert werden (hier bedeutet A die erste Speicheradresse und B die Dezimalzahl im Bereich von 0 bis 65 535):

```
100 PEEK(A) + 256 * PEEK(A+1)

200 POKE A.B-256 * INT(B/256)
210 POKE A+1,INT(B/256)
```

Zeilennummer in Komplementärdarstellung

Einige Computer speichern Adressen oder Zeilennummern über 32 767 als Negativzahlen. In diesem Fall kann die Umrechnung mit dem folgenden Befehl erfolgen:

```
100 A = A + 65536 * (A > 32767)
```

Wenn die als Variable A eingegebene Zahl größer als 32 767 ist, ist der Vergleich in der Klammer wahr, und es wird dort bei dem verwendeten Computer −1 erzeugt. Die neue Zahl wird dann durch Subtraktion von 65 336 gebildet. Ist der Vergleich in der Klammer falsch, wird dort 0 erzeugt, und der Ausdruck reduziert sich zu A = A. Die Adresse ändert sich also nicht.

Ändern einzelner Bit im Speicher

Bei einigen BASIC-Dialekten ist es notwendig, für gewisse Aufgaben einzelne Bit im Speicher zu ändern. Hierfür können OR- und AND-Verknüpfungen benutzt werden. Abb. 16 zeigt die Änderung der Bitmuster bei folgenden Befehlen:

```
100 POKE A,B OR PEEK(A)

200 POKE A,NOT B AND PEEK(A)
```

	7	6	5	4	3	2	1	0	Bitnummer
148	1	0	0	1	0	1	0	0	Ausgangsmuster in Adr. A
8	0	0	0	0	1	0	0	0	Maske mit 1 in Bit Nr. 3 (B)
156	1	0	0	1	1	1	0	0	Ergebnis

Bit Nr. 3 auf 1 setzen

156	1	0	0	1	1	1	0	0	Ausgangsmuster in Adr. A
				AND					
247	1	1	1	1	0	1	1	1	Maske mit 0 in Bit Nr. 3 (NOT B)
148	1	0	0	1	0	1	0	0	Ergebnis

Bit Nr. 3 löschen

Abb. 16: Änderung einzelner Bit im Speicher
Unterschiedliche Masken, wenn eine OR-Verknüpfung für das Setzen und eine AND-Verknüpfung für das Löschen verwendet wird

Es ist zu erkennen, daß innerhalb des POKE-Befehls der alte Inhalt der Adresse mit PEEK gelesen und dann logisch mit dem Bitmuster verknüpft wird, in dem nur das oder die zu setzenden oder zu löschenden Bits gesetzt sind.

Manchmal wird auch eine etwas andere Methode angewendet.

```
100 POKE A,(PEEK(A) AND M) + B
```

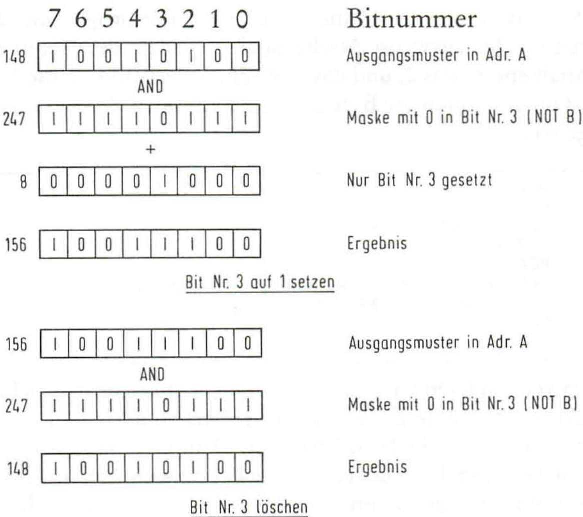


Abb. 17: Andere Methode zum Ändern einzelner Bit im Speicher
Wenn eine Maske verwendet wird, in der nur das zu bearbeitende Bit auf 1 gesetzt ist, kann es damit gelöscht werden. Das Setzen erfolgt dann durch Addition

Hier erfolgt die AND-Verknüpfung in jedem Fall durch Bitmuster mit einer Maske, in der alle nicht zu ändernden Bit gesetzt sind. Nur die zu bearbeitenden Bit enthalten eine 0. Die Maske kann auch durch **NOT B** erzeugt werden. Hiermit werden die zu bearbeitenden Bit gelöscht. Sollen sie gesetzt werden, erfolgt eine Addition mit dem Bitmuster, das nur diese Bit als Einsen enthält.

In der Literatur werden auch folgende Formulierungen empfohlen:

Setzen **POKE adr,PEEK(adr) OR 2∧ bitnummer**
Löschen **POKE adr,PEEK(adr) AND 255−2∧ bitnummer.**

7.6 Verschiedenes

Anfertigung einer Hardcopy

Diverse BASIC-Dialekte gestatten es, durch einfache Tastenbetätigung auf dem Drucker eine Kopie des Bildschirms anzufertigen. Wenn die verwendete BASIC-Version die VARPTR-Funktion kennt, gibt es eine interessante Routine, die das gleiche bewirkt.

Die VARPTR-Funktion liefert die Adresse des 4. Byte des Variablenspeichers, das hier die Länge der gespeicherten Zeichenkette enthält. Das folgende kleine Demonstrationsprogramm zeigt zunächst die Arbeitsweise. Nacheinander werden die Kennung für den Variablentyp, das 2. und das 1. Zeichen des Namens, die Länge und das untere und obere Byte des Zeigers auf den Zeichenspeicher ausgegeben.

```
V$=""
READY
>PRINT VARPTR(V$)
 27211
READY
>FOR K= -3 TO 2: PRINT PEEK(27211+K);: NEXT
 3  0  86  0  234  255
READY
>
```

Bei dem verwendeten Computer beginnt der Bildschirmspeicher in der Adresse 15 360. Je Zeile werden 64 Zeichen dargestellt, so daß die letzte (16.) Zeile in 16 320 beginnt. Damit der Speicher eingerichtet wird, enthält Zeile 1000 eine Leerzuweisung. Im 5. und 6. Byte steht die Adresse im Zeichenspeicher, von der ab die Zeichen beim Aufruf der betreffenden Zeichenkettenvariablen vom Computer zu holen sind.

Im Programm wird nun zunächst in Zeile 1020 der Wert 64 in das 4. Byte (das sonst die Zeichenkettenlänge enthält) geschrieben. Dann werden in einer Schleife jeweils die Zeichenspeicheradressen durch die Anfangsadressen einer Bildschirmzeile ersetzt, wodurch V$ jeweils den Inhalt einer Bildschirmzeile enthält und ausgedruckt werden kann.

```
10 REM HARDCOPY GENIE I    HARDCOPY
1000 V$=""
1010 V=VARPTR(V$)
1020 POKE V,64
1030 FOR K=15360 TO 16320 STEP 64
1040 POKE V+1,K AND 255
1050 POKE V+2,INT(K/256)
1060 LPRINT V$
1070 NEXT
```

Zufallszahlen eines Bereichs mit RND

Eine Reihe von BASIC-Dialekten erlaubt die einfache Erzeugung ganzzahliger Zufallszahlen im Bereich von 1 bis zu einer als Argument in die RND-Funktion einzusetzenden Zahl. Wenn diese Möglichkeit nicht gegeben ist und nur Zufallszahlen zwischen 0 und 1 erzeugt werden können, hilft folgende Anweisung, die Ganzzahlen im Bereich von 1 bis Z erzeugt:

```
100 PRINT INT(Z*RND(0))+1
```

Sollen die Zufallszahlen in einem bestimmten Bereich liegen mit K als kleinster und G als größter Zahl, kann wie folgt programmiert werden:

```
100 PRINT INT((G-K+1)*RND(0))+K:
```

Hexadezimal-Dezimal-Wandlung

Es gibt eine ganze Reihe von Methoden, Zahlen eines Systems in ein anderes zu wandeln. Das hier gezeigte Programm verwendet für die Hex-Dez-Wandlung die Reihenfolge der Zeichen im ASCII-Code.

```
10 REM Hex-Dez-Wandlung HEXDEZWA
90 INPUT"Hex-Zahl";H$
100 L=LEN(H$): D=0
110 FOR K = 1 TO L
120 A=ASC(MID$(H$,K,1))-48
130 IF A>16 THEN A=A-7
140 D=D+A*16Ä(L-K)
150 NEXT
160 PRINT D
```

In einer Schleife wird vom ASCII-Wert jeder Hex-Stelle 48 subtrahiert und das Ergebnis der Hilfsvariablen A zugewiesen. Bei den Hex-Ziffern 0 bis 9 nimmt A auch den Wert 0 bis 9 an. Danach wird A dahingehend geprüft, ob es größer als 16 ist, denn dann muß es sich um eines der Buchstabensymbole A bis F handeln. In diesem Fall wird 7 subtrahiert, so daß A schließlich den Dezimalwert des Buchstabensymbols enthält. In Zeile 140 erfolgt in üblicher Weise die Zusammensetzung der Dezimalzahl aus den Dezimalwerten der einzelnen Stelle, multipliziert mit den Potenzen von 2. Zum Verständnis dieses Programms sollte die ASCII-Tabelle in Abb. 10 auf Seite 77 herangezogen werden.

8. Fremde Programme verstehen und konvertieren

Alle Programmiersprachen bieten viele Möglichkeiten, Aufgaben auf unterschiedliche Weise zu lösen. BASIC bildet hier keine Ausnahme. Die Vielfalt der Gestaltungsmöglichkeiten wiederum bedeutet, daß fremde Programme nicht immer so ohne weiteres zu verstehen sind.

Im folgenden sollen einige Hinweise gegeben werden, wie man vorgehen kann, um fremde Programme zu durchschauen. Denn Voraussetzung für das Umstellen eines Programms ist es, daß Ablauf und Sinn der Befehle des Ursprungsprogramms begriffen sind.

Bei der Umstellung, also der Konvertierung eines Programms für den eigenen Computer, kann es zusätzliche Schwierigkeiten geben. Wir wollen deshalb erklären, wann es besser ist, ein Programm neu zu schreiben, statt es zu konvertieren.

8.1 Grobanalyse fremder Programme

Wer an einem fremden Programm interessiert ist, sollte dieses zunächst eingehend untersuchen, ehe er daran geht, es über die Tastatur in seinen Computer einzugeben. Selbst wenn das Programm auf dem gleichen Computer geschrieben wurde, lohnt sich eine vorherige Grobanalyse, bei der die folgenden Punkte untersucht werden sollten.

Aufgabe und Ergebnis des Programms

Es hat wenig Sinn, sich mit einem Programm zu beschäftigen, das nicht auch für den eigenen Bedarf nützlich ist. Als erstes sollte man darum prüfen, was das Programm bewirken soll und welches Ergebnis es liefern wird.

Während der Programmzweck meist aus der Überschrift zu ersehen ist, kann man leider nicht so ohne weiteres erkennen, welche Resultate ein Programm vorlegen wird. Nur selten findet man mit der Programmliste zusammen auch Ausdrucke des Programmlaufs mitveröffentlicht. Vielleicht liegt dies daran, daß die Erstellung einer Hardcopy, also eines Ausdrucks der Bildschirmausgabe, bei vielen Computern schwierig zu programmieren ist.

Oft kann man aber im Programm selbst an den PRINT-Befehlen erkennen, was als Ergebnis eines Programmlaufs ausgegeben wird. In jedem Fall läßt sich schnell herauslesen, ob das Programm zum Lauf einen Drucker benötigt und mit welcher Speicherung gearbeitet wird.

Wenig Sinn hat es, sich mit einem Programm zu befassen, dessen Grundproblem man nicht beherrscht. Wenn man die in einem Programm verwendeten mathematischen Rechenverfahren nicht beherrscht, versteht man es nicht. Das gleiche gilt für Spielprogramme. Man wird sie nur begreifen, wenn die Spielregeln bekannt sind oder im Programm erklärt werden.

Verwendete BASIC-Version

Nicht immer gibt der Begleittext eines Programms an, welcher BASIC-Dialekt verwendet wurde, beziehungsweise auf welchem Computer es entwickelt wurde. Da dies aber für das weitere Vorgehen wichtig ist, sollte man nach typischen Zeichen suchen, die auf einen bestimmten Computertyp schließen lassen.

Atari-Programme kann man oft daran erkennen, daß das als Kürzel statt des PRINT-Befehls eingegebene Fragezeichen im Listing stehenbleibt und nicht, wie sonst üblich, übersetzt wird. Auch das Befehlswort POSITION ist typisch für diesen Computer.

Typisch für Apple-Programme ist die Verwendung der Befehlswörter REVERSE, NORMAL, VTAB und HTAB. Nach GET folgt kein IF...THEN-Befehl, da der Computer wartet.

Commodore-Programme zeigen oft innerhalb von Zeichenketten den negativen Ausdruck gewisser Steuerzeichen. Desgleichen kann auch die häufige Verwendung der CHR$-Funktion für diesen Computer kennzeichnend sein.

In H-P-BASIC geschriebene Programme sind leicht an den sonst nicht üblichen Befehlswörtern wie DISP, SHORT, INTEGER, REAL, CURSOR, IMAGE usw. zu erkennen.

Programme, die auf dem Genie oder dem TRS 80 entwickelt wurden, haben fast immer den CLEAR-Befehl am Anfang zur Reservierung von Speicherplatz für Zeichenketten. Man findet auch manchmal den sonst nicht üblichen PRINT@-Befehl.

Ebenfalls nicht zu verwechseln sind SINCLAIR-Programme. Sie enthalten oft den Befehl PRINT AT und nach Funktionen das Argument immer ohne Klammern. LET ist stets ausgeschrieben.

Fast gleichlautend sind die Programme, die in neuen Microsoft-BASIC-Versionen geschrieben wurden, wie sie z. B. auf dem IBM-PC, dem CASIO, dem EPSON und vielen anderen Computern laufen.

8.2 Umstellen oder neu schreiben?

Manchmal dürfte es einfacher sein, eine vorliegende Programmaufgabe durch ein selbst entwickeltes Programm zu lösen, als ein fremdes Programm so umzustellen, daß es auf dem eigenen Computer läuft. Insbesondere können umständlich oder unstrukturiert geschriebene Programme auch durch Konvertierung nicht verbessert werden.

Spaghettiprogramme

Wenn ein Programm durch zu viele Sprungbefehle so unübersichtlich erscheint, daß keine klaren Module zu erkennen sind, sollte man nicht unnötige Mühe in die Entzifferung stecken. Vielfach sind bei derartigen Programmen »Rucksäcke« programmiert, worunter man durch Sprungbefehle angehängte Befehlsfolgen versteht, die ursprünglich vergessen wurden.

POKE- und PEEK-Programme

Bei einigen Computern sind POKE-Befehle für den Aufruf gewisser Programmfunktionen nicht zu vermeiden. Wer sich mit dem betreffenden Computer beschäftigt hat, kann diese auch richtig interpretieren. Wenn aber die Verwendung von POKE und PEEK in einem Programm zu häufig geschieht, ist das für Außenstehende kaum noch nachvollziehbar.

Programme mit Maschinenroutinen

Programme, in denen in Maschinensprache geschriebene Routinen verwendet oder angerufen werden, sind nur verständlich, wenn die Wirkung dieser Routinen bekannt ist. Wer kein Experte ist, sollte nicht versuchen, derartige Programme zu analysieren und zu verstehen.

Generierte Programme

Es gibt heute bestimmte unter dem Namen *Programmgenerator* oder ähnlich angebotene Hilfsprogramme, mit denen man ohne Beherrschung einer Programmiersprache individuelle BASIC-Programme erstellen kann. Die so entstandenen Programme sind nur schwer zu analysieren und kaum auf einen anderen Computertyp umstellbar.

Grafikprogramme

Obwohl die Befehlswörter bei den Grafikbefehlen oft gleich lauten, kann die Wirkung in den einzelnen Dialekten recht unterschiedlich ausfallen. Die Umstellung eines Programms mit vielen derartigen Befehlen ist darum sehr mühsam und kaum sinnvoll.

Programme mit Datenspeicherung

Erhebliche Schwierigkeiten bei der Umstellung dürften sich ergeben, wenn das fremde Programm mit Datenspeicherung auf Disketten arbeitet und einem selbst nur ein Kassettenrecorder zur Verfügung steht. Eine Umstellung ist nicht möglich, wenn im Diskettenbetrieb Daten mitten im Programm gelesen werden. Nur wenn das Einlesen aller Daten am Anfang erfolgt und während des Laufs lediglich mit dem Speicherinhalt gearbeitet wird, kann eine Umstellung in Betracht gezogen werden.

Programme der gleichen BASIC-Familie

Verhältnismäßig leicht umstellbar sind Programme, die nur in einer abweichenden Version eines auch auf dem eigenen Computer laufenden Dialekts geschrieben sind. So unterscheiden sich die auf verschiedenen Commodore-Computern laufenden BASIC-Dialekte nur geringfügig, z. B. bei den Adressen des Bildschirmspeichers. Das gleiche gilt für die verschiedenen Versionen des Microsoft-BASIC, die auch nur minimale Abweichungen aufweisen.

Microsoft-BASIC und andere Dialekte

Da POKE-Befehle bei Programmen, die in Microsoft-BASIC geschrieben wurden, eigentlich nicht notwendig sind, lassen sich Microsoft-BASIC-Listings gut in andere Dialekte umschreiben. Auf der anderen Seite sind Programme, die auf Computern mit häufigem Zugriff auf die Maschinensprache (z. B. Acorn, Atari, Commodore u. a.) entwickelt wurden, nur schwer konvertierbar. Wenn aber abweichende Befehlswörter mit im wesentlichen gleicher Befehlswirkung im fremden Programm vorkommen, ist die Umstellung möglich.

8.3 Fremde Programme untersuchen

Wer sich mit fremden Programmen befaßt, sei es, um daraus zu lernen, sei es, um sie in abgewandelter Form auf seinem Computer zum Laufen zu bringen, muß diese verstehen. Hierfür ist eine eingehende Untersuchung des Programmaufbaus und des Programmablaufs erforderlich, für die im folgenden einige Hinweise gegeben werden.

Aufgliederung in Programm-Module

Ein vernünftig geschriebenes Programm muß strukturiert sein, das heißt, es muß aus in sich abgeschlossenen Modulen bestehen, die nur einen Eingang und (möglichst) auch nur einen Ausgang haben sollten. Es gibt Programm-Autoren, die die einzelnen Programmteile durch Kommentar- oder Leerzeilen optisch gut voneinander trennen.

Bei anderen Programmen ist die Gliederung nicht immer so deutlich zu erkennen. Dies braucht nicht zu bedeuten, daß der Verfasser des Programms nichts von der Strukturierung hält. Eine deutliche Abgrenzung der Strukturblöcke wird zum Beispiel dadurch vorgenommen, daß diese immer in geraden Hunderter- oder Tausender-Zeilennummern beginnen.

Der erste Schritt einer Programmanalyse besteht also darin, den Aufbau des Programms zu erforschen und gegebenenfalls den Beginn und das Ende einzelner Programm-Module herauszuarbeiten. Hierbei kann man wie folgt vorgehen:

- Durch waagerechte Striche wird jeweils der Anfang und das Ende eines Moduls markiert, wenn nicht schon eine Markierung im Programm steht.
- Sehr zweckmäßig ist es, alle FOR...NEXT-Schleifen optisch zu kennzeichnen und dabei auch auf Schleifenverschachtelungen zu achten.
- Durch Unterstreichen markiert man alle in GOTO und GOSUB stehenden Zeilennummern.
- Hierbei erkennt man, wo Unterprogramme anfangen. Sie werden am Rand in ganzer Länge bis zum RETURN-Befehl mit zwei senkrechten Strichen gekennzeichnet.
- Wichtig ist aber auch die klare Kennzeichnung aller Zeilen, die Ziel eines Sprungbefehls sind. Hierfür kann beispielsweise ein kleiner Pfeil verwendet werden.

Abb. 18 zeigt ein nach diesem Verfahren markiertes Programm.

```
10 REM Stichwörter einsortieren STICHWO2
40 CLEAR 10000
50 DIM X$(500).H(500)
60 INPUT"Altes Verzeichnis einlesen <J/N>":Y$
70 IF Y$="j" OR Y$="J" THEN 100
80 IF Y$="n" OR Y$="N" THEN 200
90 GOTO 70
100 OPEN"I".#1."DADIDA"
110 INPUT#1.Z
120 FOR K = 0 TO Z
130 INPUT#1.X$(K).H(K)
140 NEXT
150 CLOSE
200 PRINT"Stichwörter eingeben. mit 0 beenden"
210 INPUT X$
220 IF X$ ="0" THEN 400
230 Z = Z + 1
240 B$ = LEFT$(X$.1)
250 IF ASC(B$) >= 97 THEN 290
270 X = ASC(B$) + 32
280 X$ = CHR$(X) + RIGHT$(X$.LEN(X$)-1)
290 X$(Z)=X$: H(Z)=Z
300 FOR K = Z TO 1 STEP -1
310 IF X$(H(K)) > X$(H(K-1)) THEN 210
320 Y=H(K):H(K)=H(K-1):H(K-1)=Y
330 NEXT K
340 GOTO 210
400 OPEN"O".#1."DADIDA"
410 PRINT#1.Z
420 FOR K = 0 TO Z
430 PRINT#1.X$(K);".";H(K)
440 NEXT
450 CLOSE
500 FOR K = 0 TO Z
510 PRINT X$(H(K)).
520 NEXT
```

Abb. 18: Programm »Stichwörter einsortieren« mit Markierung

Erkennen des Programmablaufs

Wenn das Programm so weit markiert ist, sollte man versuchen zu erkennen, wie das Programm abläuft. Hierbei ist zunächst zu ergründen, auf welche Weise und in welcher Reihenfolge die Programm-Module miteinander verknüpft sind.

Am besten läßt sich dies in einem groben *Flußdiagramm,* das auch *Programm-Ablaufplan* genannt wird, darstellen. Sprünge, die zu Schleifen innerhalb eines Moduls gehören, durch die also keine Änderung des Programmlaufs erfolgt, brauchen hierbei nicht berücksichtigt zu werden. Es sind aber alle Aussprünge aus einem Modul zu erfassen.

Nur bei kleinen Programmen ist es möglich, den Programmlauf durch Pfeile in der Programmliste zu markieren. Wem das Zeichnen eines Flußdiagramms zu mühsam ist, der kann sich manchmal mit einer Zeilen-Ablaufliste, auf der die Zeilennummern aller

Sprungbefehle und aller Sprungziele stehen, begnügen. Eine derartige Liste mag auch bei größeren Programmen dazu dienen, einen ersten Überblick über den Programmlauf zu bekommen. Abb. 19 zeigt, wie der Programmablauf des Programms in Abb. 18 als Zeilen-Ablaufliste dargestellt werden kann.

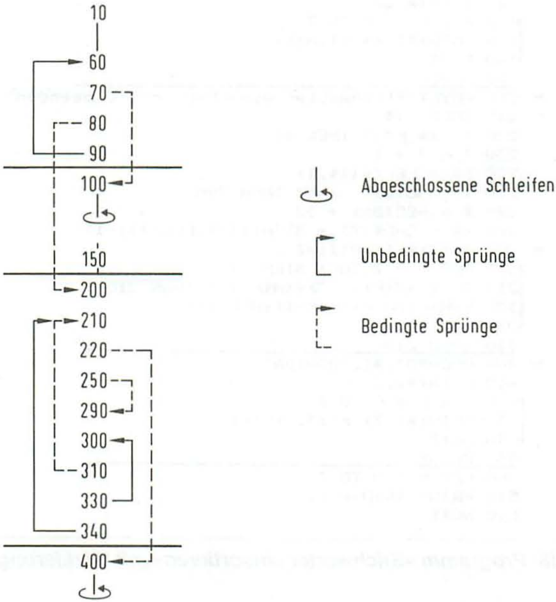


Abb. 19: Zeilenablaufliste für das Programm in Abb. 18
Bei dieser Darstellung werden nur die Zeilennummern aufgeführt, die einen Sprungbefehl oder ein Sprungziel enthalten. Abgeschlossene Schleifen brauchen nicht genau bezeichnet zu werden

Besondere Aufmerksamkeit muß man hierbei den Unterprogrammen widmen. Viele Programmierer verwenden GOSUB-Befehle, wenn auch einfache GOTO-Befehle gereicht hätten. Letztere genügen immer dann, wenn das Programm nach jedem Anruf an der gleichen Stelle weitergehen soll. Bei der Analyse von Programmen mit vielen GOSUB-Befehlen besteht die Schwierigkeit, daß man übersieht, wo das Programm nach dem Unterprogramm fortgesetzt wird. Die Rückkehrstellen nach Unterprogramm-Aufrufen müssen also auch klar aus dem Ablaufplan ersichtlich sein.

Wer ein Programm so weit analysiert hat, wird oft schon recht gut erkennen, was in dem Programm geschieht. Wenn die im Programm verwendeten Befehle auch auf dem eigenen Computer laufen, kann man eintippen. Gegebenenfalls ändere man einzelne Befehle. Will man das Programm darüber hinaus bis zur letzten Befehlsfolge verstehen, muß man meist noch mehr Mühe aufwenden.

8.4 Analyse einzelner Befehlsfolgen

Ist der Ablauf eines Programms herausgearbeitet, so bedeutet dies längst nicht immer, daß auch alle Befehle begriffen sind. Hier tut oft noch eine detaillierte Analyse not, bei der man logisch vorgehen und gegebenenfalls den Computer als Hilfe benutzen muß.

Aufstellung der Variablenliste

Wenn im Programm selbst oder in der Beschreibung keine Variablenliste zu finden ist, muß als erstes diese angelegt werden. Die Zusammenstellung einer solchen Liste kann etwas Detektivarbeit erfordern. Am besten fängt man damit an, die Variablennamen aufzuschreiben, die in INPUT- oder GET-Befehlen vorkommen; denn diese werden meistens im Befehl selbst oder in einem PRINT-Befehl davor erklärt.

Auch die Ausgabe-Befehle können bei der Aufstellung der Variablenliste hilfreich sein, da im Programm für den Anwender meist erklärt wird, was Ausgaben bedeuten. Erfolgt die Ausgabe in Tabellenform, muß man die Kopfzeile der Tabelle betrachten.

Schwieriger ist es, die Bedeutung der Variablen zu ermitteln, die im Programmlauf als Zwischenspeicher usw. gebraucht werden. Man sollte sie anfangs darum nur mit ihrem Namen in die Liste aufnehmen und die Bedeutung erst nachtragen, wenn man die Befehle, in denen sie vorkommen, im einzelnen untersucht hat.

Bei der Variablenliste gilt es auch daran zu denken, daß eine Variable im Programm durchaus verschiedene Bedeutungen haben kann. Viele Programmierer verwenden bewußt einen schon vorher benutzten Variablennamen, sofern die alte Bedeutung im Programm nicht mehr gebraucht wird. Man erinnere sich hier nur daran, daß für neue FOR...NEXT-Schleifen immer wieder die gleiche Variable als Laufvariable verwendet werden kann, wenn die Schleifen, in denen sie vorher gebraucht wurde, abgeschlossen sind.

Vor Aufstellung der Variablenliste untersucht man am besten, ob es am Anfang des Programms oder an anderer Stelle Definitionsbefehle gibt, die bestimmte Namen an bestimmte Variablentypen binden. Bei einigen BASIC-Dialekten ist das erlaubt.

Besonders zu untersuchen sind Feldvariable, bei denen man herausfinden muß, welche Bedeutung die Feldinhalte wie auch die Indices haben.

Nicht sofort verständliche Befehlsfolgen

Wenn man so weit gekommen ist, wird der Aufbau der meisten Programmzeilen einigermaßen verständlich sein, aber oft wird man auch auf Befehlsfolgen stoßen, deren Sinn nicht sofort einleuchtet.

Schwierigkeiten kann es geben, wenn die *CHR$-Funktion* einbezogen ist. Falls das verwendete BASIC den Standard-ASCII-Code benutzt, hilft ein Blick auf eine ASCII-Tabelle. Leider gibt es aber auch Dialekte, die einen eigenen Code heranziehen. Da dieser aber fast immer, zumindest in der Reihenfolge der Zeichen, an den ASCII-Code angelehnt ist, hilft dann auch einiges Nachdenken. Vielfach werden Cursorbewegungen mit PRINT CHR$(...) erzeugt.

Über *POKE-Befehle* wurde schon einiges gesagt. Oft dienen sie nur zur Bildschirmsteuerung, z. B. zum Einschalten der negativen Darstellung oder zur Wahl einer besonderen Betriebsart. Man versucht dann, ob das Programm auch ohne diese Befehle läuft.

Schwer verständlich können ungewöhnliche *Vergleichformulierungen* sein, über die einiges im Abschnitt 7.3 gesagt wurde. Insbesondere ist die Wirkung des NOT-Operators manchmal nicht auf den ersten Blick zu erkennen. Es ist darum empfehlenswert, unklare Vergleiche einfach im Direktbetrieb auf dem Computer auszuprobieren.

Auch die von manchen Programm-Autoren sehr gern verwendeten *Funktionsdefinitionen mit DEF FN* sind hin und wieder nicht sofort zu durchschauen, da sich die Syntaxregeln bei den einzelnen Dialekten unterscheiden. Hier hilft nur eine sorgfältige Untersuchung des Programms und gegebenenfalls das Aufschreiben der Funktionszuweisung, damit man letztere, wenn sie aufgerufen wird, immer zur Hand hat.

Bei mehrfach *verschachtelten Funktionen* muß man immer überlegen, wie der Computer sie verarbeitet. Als erstes wird stets der Wert der innersten Klammer bestimmt.

Schleifenläufe analysieren

Mit am schwersten zu verstehen sind Schleifenläufe, in denen auch die Indices von Feldvariablen verändert werden, wie es zum Beispiel bei vielen Sortierprogrammen der Fall ist. Wenn derartige Befehlsfolgen nicht durchschaubar sind, sollte man den Computer zu Hilfe rufen. Da es sich meist nur um wenige Befehle handelt, kann man sie als kleines Testprogramm eingeben und ausprobieren, wie die einzelnen Variablen durch das Programm verändert werden.

```
10 REM Bubblesort
20 REM Zu sortierendes Feld in A(x)
1000 N = 5
1010 FOR K = 2 TO N
1020 FOR J = N TO K STEP -1
1030 IF A(J-1) > A(J) THEN 1050
1040 GOTO 1070
1050 Y=A(J): A(J)=A(J-1): A(J-1)=Y
1070 NEXT J
1080 NEXT K
1090 PRINT

30 FOR K = 1 TO 5
40 A(K) = 10-K
50 PRINT A(K);: NEXT K
60 PRINT:PRINT

2000 FOR K=1 TO 5
2010 PRINT A(K);
2020 NEXT
1051 PRINT K;J,A(1);A(2);A(3);A(4);A(5)

RUN
 9  8  7  6  5

 2  5               9  8  7  5  6
 2  4               9  8  5  7  6
 2  3               9  5  8  7  6
 2  2               5  9  8  7  6
 3  5               5  9  8  6  7
 3  4               5  9  6  8  7
 3  3               5  6  9  8  7
 4  5               5  6  9  7  8
 4  4               5  6  7  9  8
 5  5               5  6  7  8  9

 5  6  7  8  9
READY
```

Abb. 20: Analyse eines Programms mit verschachtelten Schleifen
Da derartige Programme oft schwer zu durchschauen sind, wird im Programm an geeigneter Stelle ein PRINT-Befehl eingesetzt, der alle Variablen bei jedem Schleifenlauf ausgibt. Dadurch sind Veränderungen gut zu erkennen. Da es sich hier um ein Sortierprogramm handelt, muß vorher noch die unsortierte Folge eindeutig bestimmt und nachher das Ergebnis ausgegeben werden

Hierbei gehe man aber gezielt vor und beachte folgende Hinweise:

- Vor den zu testenden Programmteil sind einige Befehle zu schreiben, in denen den verwendeten Variablen eindeutige Werte zugewiesen werden.
- Ebenso müssen am Ende einige Befehle stehen, die das Ergebnis der Schleifenläufe zeigen.
- Gegebenenfalls ist die Anzahl der Schleifenläufe, soweit sinnvoll, zu begrenzen, z. B. durch entsprechende Voreinstellung im FOR-Befehl.
- Um zu sehen, was in jeder Schleife passiert, kann dort ein provisorischer STOP-Befehl eingebaut werden. Man kann dann nach dem Anhalten des Programms im Direktbetrieb die Variableninhalte ausgeben lassen.
- Sehr zweckmäßig ist es auch, direkt in die Schleifen zusätzliche PRINT-Befehle einzusetzen, die die Werte der in der Schleife geänderten Variablen ausgeben.
- Wenn in der Schleife der Index einer Feldvariablen verändert wird, muß auch die dazugehörige Variable selbst gedruckt werden.
- Zusätzliche Testbefehle sollten niemals an eine Befehlszeile angehängt werden, sondern stets in einer neuen Zeilennummer stehen. Empfehlenswert ist es hierfür zum Beispiel, immer Zeilennummern, die mit einer 1 enden, zu verwenden. Man erkennt die Zusatzbefehle dann schnell im Listing und kann sie nach der Prüfung wieder löschen.
- Wenn in der Schleife auch Ausgaben stehen, kann man in den Testbefehlen nach PRINT ein Komma setzen, um die Testausgaben auf dem Bildschirm von den normalen Ausgaben abzuheben.

Programmprüfung im Trace-Betrieb

Der von vielen BASIC-Dialekten angebotene *Trace-Betrieb*, bei dem alle vom Programm durchlaufenen Zeilen auf dem Bildschirm gezeigt werden, ist nur begrenzt zur Prüfung eines Programmablaufs brauchbar. Das unkontrollierte Einsetzen des TRACE-Befehls in ein Programm führt nur zu einem Zahlenwirrwarr auf dem Bildschirm. Wenn der Programmlauf hiermit geprüft werden soll, ist die Beachtung der folgenden Hinweise zweckmäßig:

- TRACE nur dort in das Programm schreiben, wo der Ablauf unklar ist – und sofort danach mit TROFF wieder aufheben.
- Damit die Ausgabe der im Trace-Betrieb angezeigten Zeilen-

nummern sich von den im Programm geforderten Ausgaben abhebt, muß vor die im Programm stehenden PRINT-Befehle ein leerer PRINT-Befehl gesetzt werden.

8.5 Beispiele für Programmanalysen

Wieweit es notwendig ist, ein Programm zum Verständnis vollständig zu analysieren, hängt natürlich vom Kenntnisstand des Benutzers ab. Im folgenden wird an zwei kleinen Beispielen gezeigt, wie ein Programm untersucht werden kann, wenn der Programmablauf beim bloßen Durchsehen nicht klar geworden ist.

Programm: »Stichwörter einsortieren« (bezieht sich auf Seite 179)
Wie der Titel sagt, handelt es sich um ein Programm, das Stichwörter sortiert. Dabei kann davon ausgegangen werden, daß die Ordnung nach dem Alphabet erfolgt. Wie dies abläuft, ist aber nicht zu erkennen.

Auf welche Weise die Markierung des Programms geschehen kann, ist bereits in der Abb. 18 gezeigt. Man erkennt, daß im Programm fünf jeweils mit geraden 100er-Nummern beginnende Module stehen.

1. Dimensionierung und Wahl des Programmlaufs
2. Einlesen des alten Stichwortverzeichnisses von der Diskette
3. Eingeben und Sortieren der Stichwörter
4. Abspeichern des sortierten Verzeichnisses
5. Ausgeben der sortierten Stichwörter.

Man ersieht im Initialisierungsmodul am Befehl CLEAR 10 000, daß das Programm für ein Genie oder einen TRS 80 geschrieben wurde. Das 2. und das 4. Modul weisen aus, daß mit Diskettenspeicherung gearbeitet wird. Da die Speicherung aber am Beginn beziehungsweise Ende des eigentlichen Sortiermoduls erfolgt, könnte auch auf Kassettenspeicherung umgestellt werden.

Die danach aufgestellte Zeilen-Ablaufliste in Abb. 19, Seite 180, läßt erkennen, daß beim ersten Anlegen des Stichwortverzeichnisses das Modul zum Einlesen des alten Verzeichnisses übersprungen wird. Im übrigen laufen dann die Module sequentiell, also in der Reihenfolge, in der sie im Programm stehen, hintereinander ab.

Wenn die Arbeitsweise des Programms jetzt noch nicht klar ist, müssen die Module im einzelnen untersucht werden. Dabei ist eine Variablenliste anzulegen, die – allerdings erst am Ende der Untersuchung – wie folgt aussieht:

Z	Zähler für Stichwörter
K	Schleifenzähler
X	Hilfsvariable
X$	eingegebenes Stichwort
Y$	Eingabevariable
B$	erste Stelle eines Stichwortes
X$(x)	Feld für Stichwörter
H(x)	Indexfeld für Stichwortfeld.

Im ersten Modul wird ein Zeichenketten- und ein Zahlenfeld dimensioniert. Wofür diese verwendet werden, ist erst im zweiten Modul zu erkennen. Da die Stichwörter in Feld X$(x) mit dem gleichen Index wie im Feld H(x) laufen, ist zu vermuten, daß letzteres nur als Indexfeld dient. Die Variable Z dient als Zähler für die Stichwörter, denn durch sie wird die Leseschleife begrenzt. Sie wird darum auch als erste von der Datei gelesen.

Im dritten Modul steht der Eingabebefehl für die Stichwörter, und man erkennt, wie die Eingabe zu beenden ist, und daß bei jeder Eingabe der Zähler erhöht wird.

In den folgenden Befehlen wird das erste Zeichen vom Eingabestring abgetrennt und geprüft, ob es einen größeren ASCII-Wert als 96 hat. Ein Blick auf die Codetabelle zeigt, daß es sich dann um einen Kleinbuchstaben handeln muß. In diesem Fall erfolgt ein Sprung nach Zeile 290, wo der Eingabestring dem Element des Feldes X(x) mit dem Index Z zugewiesen wird. Zugleich wird auch Z noch einmal im gleichen Element des Indexfeldes H(x) abgelegt.

Wenn als erstes ein Großbuchstabe gefunden wurde, wird 32 – also der Unterschied zwischen Groß- und Kleinbuchstaben – im ASCII-Code hinzuaddiert und dann der entsprechende Kleinbuchstabe vorn an die anderen Zeichen des Eingabestrings gesetzt. So können beliebig abwechselnd mit großen oder kleinen Anfangsbuchstaben eingegebene Stichwörter einheitlich nach dem Alphabet sortiert werden.

Da das Programm nach Zeile 290 gleich mit Zeile 300 weiterläuft, ist zu erkennen, daß nach jeder Eingabe sortiert wird. Dies geschieht in der Weise, daß das neue Stichwort, am Ende des Feldes beginnend, in einer Schleife jeweils mit dem vorstehenden Element verglichen wird. Ist dieses größer als das neue Element, heißt dies, daß es im Alphabet vor diesem einzuordnen ist, und die Sortierschleife wird abgebrochen.

Umgekehrt wird, wenn das neue Element nicht größer als das vorherstehende ist, nur das Indexfeld umgestellt, was schneller geht als die Umstellung der Elemente eines Zeichenkettenfeldes.

Im vierten Modul wird nach Beendigung der Eingabe eine sequentielle Datei geöffnet, und es wird dort zuerst der Zählerstand, dann werden die Elemente beider Felder gespeichert.

Im Ausgabemodul werden die Stichwörter in der Reihenfolge der Sortierung im Indexfeld ausgegeben.

Programm: »Aufteilung in Gruppen«

Bei dem in Abb. 21 gezeigten Programm ist nicht gleich auf den ersten Blick zu erkennen, welchem Zweck es dient. Die Analyse wird darum auch wieder mit der Markierung der Programm-Module begonnen.

1. Initialisierung und Abfrage
2. Einordnung eines Teilnehmers in eine Gruppe
3. Einordnung eines Teilnehmers in eine Gruppe
4. Ausgabe der Gruppeneinteilung.

```
10 REM Aufteilung in Gruppen  AUFTEIL 627 Byte
20 CLS: INPUT"Teilnehmerzahl";T
30 INPUT"Wieviele Gruppen";G
40 TG=INT(T/G):
50 DIM G$(G+1,TG+1)
100 FOR K = 1 TO TG*G
110 INPUT"Name oder Kürzel eingeben";N$
120 N=RND(G)
130 FOR J = 1 TO TG
140 IF G$(N,J) <> "" THEN 160
150 G$(N,J)=N$: GOSUB 300: GOTO 180
160 NEXT J
170 GOTO 120
180 NEXT K
190 IF T = TG*G THEN 400
200 FOR K = TG*G+1 TO T
210 INPUT"Name oder Kürzel eingeben";N$
220 N=RND(G)
230 IF G$(N,TG+1)<>"" THEN 220
240 G$(N,TG+1)=N$: GOSUB 300
250 NEXT
260 GOTO 400
300 PRINT: PRINT"Teilnehmer ";N$;" zur Gruppe "N
310 RETURN
400 CLS: FOR K = 1 TO G STEP 2
410 PRINT"GRUPPE";K,"GRUPPE";K+1: PRINT
420 FOR J = 1 TO TG+1
430 PRINT G$(K,J), G$(K+1,J)
440 NEXT J
450 NEXT K: END
```

Abb. 21: Programm »Aufteilung in Gruppen« mit Markierung

Außerdem gibt es noch ein Unterprogramm (das aus den Modulen 2 und 3 angerufen wird), in dem einem Teilnehmer mitgeteilt wird, zu welcher Gruppe er gehört.

Der Programmablauf ist hier sehr einfach, da die vier Module nach Abarbeitung der darin programmierten Schleifen nacheinander laufen.

Da auch hier der Lauf der einzelnen Module nicht sofort klar ist, wird eine Variablenliste angelegt:

T	Teilnehmerzahl, die in Gruppen aufgeteilt wird
G	Anzahl der Gruppen
TG	Teilnehmer pro Gruppe ohne Überhang
J, K	Laufvariable
N	zufallsbedingte Gruppennummer
N$	eingegebener Teilnehmername
G$ (x,y)	zweidimensionales Feld für verteilte Teilnehmer mit
x =	Gruppennummer
y =	Nummer in der Gruppe.

Bei der Analyse fällt im 1. Modul die besondere Art der Dimensionierung für das Feld auf. Es ist vorläufig nicht zu ermitteln, warum die Zeilen und Spalten der Matrix um den Wert 1 höher dimensioniert werden, als es Gruppen beziehungsweise Teilnehmer gibt. Hier offenbart sich aber die Bedeutung von TG. Es ist die Anzahl der Teilnehmer pro Gruppe, die dort eingeteilt werden können, wenn die Teilnehmerzahl, dividiert durch die Gruppenzahl, keinen Rest ergibt.

Im zweiten Modul ersieht man aus der RND-Funktion in Zeile 120, mit der die Gruppennummer bestimmt wird, daß dieses Programm wahrscheinlich auf einem Genie, einem TRS 80 oder einem Acorn geschrieben wurde. Denn bei diesen Computern ergibt eine ganzzahlige positive Zahl als Argument Zufallszahlen zwischen 1 und dieser Zahl.

Es laufen zwei verschachtelte Schleifen. In der äußeren werden die Teilnehmer eingeteilt, die gleichmäßig ohne Überhang auf die Gruppen verteilt werden können. In der inneren Schleife geschieht in Zeile 150 jeweils eine Prüfung, ob ein Gruppenplatz schon besetzt ist. Wenn dies der Fall ist, erfolgt die Erhöhung der Laufvariablen, und der nächste Platz wird untersucht. Hierbei läuft die Schleife maximal so oft, wie Teilnehmer ohne Überhang in eine Gruppe eingewiesen werden können.

Wenn eine Gruppe voll ist, wird eine neue Zufallszahl, also eine neue Gruppennummer, angefordert, und die innere Schleife läuft neu, bis alle Teilnehmer ohne Überhang verteilt sind. Der Befehl in Zeile 190 sorgt schließlich dafür, daß das nächste Modul übersprungen wird, wenn kein Teilnehmerüberhang vorhanden ist.

Im dritten Modul werden die Überhangteilnehmer verteilt. Da pro Gruppe nur ein Teilnehmer zugewiesen zu werden braucht, entfällt die innere Schleife. Sonst ähneln die Befehle dem Modul 2.

Im vierten Modul erfolgt die Ausgabe der den einzelnen Gruppen zugeteilten Teilnehmer. Es werden zwei verschachtelte Schleifen verwendet, die erste für die Gruppen, die zweite für die Teilnehmer. Aus Zeilen 410 und 430 ist ersichtlich, daß in 2 Spalten gedruckt wird. Jetzt kommt man auch dahinter, warum die Felder um den Wert 1 höher dimensioniert wurden. Dies ist nötig, da bei ungerader Gruppenzahl die Gruppennummer K sonst in der letzten rechten Spalte größer würde, als es die Dimensionierung zuläßt.

Diese kleinen Beispiele zeigen schon, daß es einiger Überlegungen bedarf, um ein Programm zu analysieren, sofern es alle Möglichkeiten ausnutzt, die BASIC bietet. Bei großen Programmen kann diese Arbeit sehr mühsam sein. Aber aus der Analyse fremder Programme läßt sich viel lernen. Das gilt im positiven Sinn, wenn diese gut geschrieben sind, aber auch negativ, wenn man in fremden Programmen erkennt, wie man es eigentlich nicht machen sollte.

Register

Verzeichnis der BASIC-Fachausdrücke

Verzeichnis der BASIC-Befehle

humboldt

Bücher, die zur Sache kommen!

Der Autor: Dr. Hans-Joachim Sacht hat sich seit der Einführung von Mikroprozessoren in Europa mit deren Technik und Anwendung befaßt. In unzähligen Seminaren machte er auch Nicht-Fachleute mit Mikrocomputern und Programmiersprachen bekannt. Er gehört zu den meistgelesenen Fachautoren.

- Sie erfahren hier, wie sich die BASIC-Versionen der wichtigsten Home- und Personal-Computer unterscheiden und wie man BASIC-Dialekte analysiert.
- In diesem Band finden Sie die BASIC-Dialektbesonderheiten von Acorn BBC und Electron, Apple IIe, Alphatronic PC, Atari, Casio PP-200, Commodore 3000/4000/8000, Commodore VC 20 und 64; Epson HX-20, Dragon, Genie und Colour Genie, Hewlett-Packard HP 86, IBM PC und IBM Compatible Computer, MSX Standard, Oric Atmos, Laser 310/2001/VZ 200, Schneider CPC 464, Sharp MZ-80 und MZ-700, Sinclair ZX 81 und Spectrum, Spectra Video SV-318 und SV-328, Tandy TRS 80, Thomson TO 7, Texas Instruments TI 99/4A ...
- Sie erhalten viele Hinweise, wie fehlende BASIC-Befehle ersetzt werden können.
- Wir erklären BASIC-Routinen, die nicht auf den ersten Blick verständlich sind, und geben Anleitungen, wie ein fremdes Programm umgestellt werden kann.

Praktische Ratgeber

ISB N 3-581-66524-7 DM +011.80

T 3-28-99

01180

9 783581 665240